Android-haittaohjelmien tunnistaminen koneoppimismenetelmin

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Pro gradu -tutkielma
Ohjaaja: Martti Juhola
Heinäkuu 2017

Sisältö

Kuvat

Taulukot

Ohjelmakoodit

Tampereen yliopisto
Luonnontieteiden tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Miika Koskela: Android-haittaohjelmien tunnistaminen koneoppimismenetelmin
Pro gradu -tutkielma, 57 sivua
Heinäkuu 2017

Haittaohjelmien ja niiden eri variaatioiden sekä (haitta)ohjelmanäytteiden päivittäisen valtavan määrän myötä manuaalinen ohjelmanäytteiden analysointi ja kategorisointi ei ole enää ajankäytöllisesti järkevää tai edes mahdollista. Koneoppimismenetelmin pyritään automatisoimaan ohjelmanäytteiden kategorisointia mahdollisimman pitkälle, jolloin haittaohjelmia analysoivat tutkijat voivat keskittyä erityisesti valittujen kohteiden yksityiskohtaisempaan tarkasteluun. Lisäksi entuudestaan tuntemattomien haittaohjelmien tunnistamiseen voidaan käyttää erilaisia koneoppimismenetelmiä.

Mobiililaitteiden yleistymisen myötä haittaohjelmien laatijat ovat ottaneet kohteekseen myös erilaiset älylaitteet, kuten älypuhelimet. Tässä tutkielmassa tarkastellaan koneoppimismenetelmien soveltamista Android-haittaohjelmien automaattiseen tunnistamiseen ja kategorisointiin. Muuttujina ovat Androidin järjestelmäoikeudet. Menetelmiksi valittiin *k*:n lähimmän naapurin luokittelumenetelmä, päätöspuu sekä satunnaismetsä. Tulokset olivat hyviä, varsinkin satunnaismetsällä.

Avainsanat ja -sanonnat: Android, järjestelmäoikeus, haittaohjelma, koneoppiminen, *k*:n lähimmän naapurin luokittelu, päätöspuu, satunnaismetsä

1 Johdanto

Monet tutkijaryhmät, kuten esimerkiksi Suarez-Tangil, J. E. Tapiador ym. (2014), Moonsamy ym. (2014) sekä Burguera ym. (2011), ovat tutkineet koneoppimismenetelmien käyttöä Android-haittaohjelmien tunnistamisessa ja kategorisoinnissa. Yksi motivoiva tekijä tähän on ollut erilaisten älylaitteille suunnattujen haittaohjelmien määrän ja niiden variaatioiden voimakas kasvu (Suarez-Tangil, J. E. Tapiador ym., 2014; Felt, Finifter ym., 2011).

Haittaohjelmien ja niiden variaatioiden alati kasvava ja valtava määrä tekee ohjelmanäytteiden manuaalisen tarkastelemisen ja kategorisoinnin käytännössä mahdottomaksi. Ohjelmanäytteiden analysointiin tarvitaan siis älykkäitä järjestelmiä, jotka voivat auttaa haittaohjelman analysoijia (Suarez-Tangil, J. E. Tapiador ym., 2014, ss. 1104-1105). Esimerkiksi F-Secure kirjoittaa hyödyntävänsä koneoppimismenetelmiä noin puolen miljoonan uuden ohjelmanäytteen automaattiseen kategorisointiin päivittäin (F-Secure, 2016c).

Lisäksi loppukäyttäjän laitteelle asennettavat haittaohjelman torjuntaan tarkoitetut ohjelmistot ovat kehittyneet ensimmäisistä, yksistään *allekirjoitukseen* (signature) perustuvista tunnistusmenetelmistä kehittyneempiin tekniikoihin, joilla voidaan tunnistaa myös entuudestaan tuntemattomia haittaohjelmia. Nykyiset, kehittyneemmät haittaohjelmien torjuntaohjelmistot yhdistelevät useita eri tekniikoita, kuten käyttäytymisen ja tiedoston tunnettuuden analysointia. Käyttäytymistä analysoimalla voidaan myös havaita ja pysäyttää entuudestaan tuntematon ohjelma, joka käyttäytyy haittaohjelmalle tyypillisellä tavalla. (F-Secure, 2016[g]; F-Secure, 2016b, ss. 7-8)

Kuten edellä todettiin, haittaohjelmien tunnistaminen perustui aluksi haittaohjelmista luotuihin allekirjoituksiin (F-Secure, 2016[g]). Allekirjoitus oli tuolloin vain haittaohjelmasta laskettu yksittäinen *hajautusarvo* (hash) tai joukko hajautusarvoja, mutta nykyään allekirjoitukseksi sanotaan myös muita, kehittyneempiä tunnisteita. (Kaspersky, 2016a; F-Secure, 2016[g]) Allekirjoitusta voidaan kutsua myös *tunnisteeksi* (detection) (F-Secure, 2016[d]).

Yksistään allekirjoitukseen perustuva tunnistusmenetelmä ei kuitenkaan kykene tunnistamaan entuudestaan tuntemattomia haittaohjelmia (F-Secure, 2016[d], katso kohdasta "Signature"). Allekirjoituksen perusteella tunnistaminen edellyttää siis sitä, että haittaohjelmasta on olemassa näyte, joka on kategorisoitu haittaohjelmaksi ja luodut allekirjoitukset on päivitetty haittaohjelmien torjuntaohjelmistoon (F-Secure, 2016[d], katso kohdasta "Detection").

Tässä tutkielmassa tutustutaan koneoppimismenetelmien käyttöön Android-haittaohjelmien tunnistamisessa. Tutkielman yhteydessä tehtiin myös käytännön tutkimustyötä ja siitä saatuja tuloksia esitellään myöhemmin luvussa 5. Tutki-

mustyö sisälsi muun muassa muuttujien valitsemisen ja muunnoksen *ominaisuusvektoreiksi* (feature vector) sekä luokittelun ja tulosten tarkastelun. Luokittelumenetelmiksi valittiin lopulta k:n lähimmän naapurin menetelmä, päätöspuu sekä satunnaismetsä, joiden toimintaperiaatteet ja käytännön sovellutus esitellään pääpiirteissään myöhemmin.

Tutkielman loppuosa on jaoteltu seuraavasti: Luvussa 2 tarkastellaan haittaohjelmia lyhyesti. Luvussa 3 tutustutaan tämän tutkielman kannalta oleellisiin Android-käyttöjärjestelmän osiin, Android-sovelluksiin sekä turvallisuuskysymyksiin. Luvussa 4 esitellään käytetty data ja muuttujat. Luvussa 5 esitellään käytetyt koneoppimismenetelmät, niiden käytännön soveltaminen sekä tulokset, joita verrataan aiempaan tutkimukseen. Lopuksi luvussa 6 on yhteenveto, jossa muun muassa vertaillaan tutkielmassa käytettyjen menetelmien antamia tuloksia.

2 Haittaohjelmat

Termillä *haittaohjelma* tarkoitetaan yleisemmin sellaista ohjelmaa, joka aiheuttaa – nimensä mukaisesti – tavalla tai toisella haittaa laitteelle ja sen käyttäjälle (Kaspersky, 2017[a]) tai omistajalle. Haittaohjelmia voidaan luokitella eri tavoin. Esimerkiksi F-Secure (F-Secure, 2016a) luokittelee ohjelmistot neljään eri kategoriaan niiden aiheuttaman riskin mukaan:

- Puhtaat (Clean)
- Mahdollisesti ei-toivotut (Potentially Unwanted Applications)
- Ei-toivotut (Unwanted Applications)
- Haitalliset (Harmful)

Haitallisiin ohjelmiin eli haittaohjelmiin kuuluvat muun muassa *virukset* (virus), *madot* (worm) ja *troijalaiset* (trojan or trojan horse). Mahdollisesti ei-toivottuihin ja ei-toivottuihin sovelluksiin kuuluvat muun muassa *vakoiluohjelmat* (spyware) ja *seurantaohjelmat* (trackware) (F-Secure, 2016a).

Haittaohjelmien vaikutukset ja toiminnot ovat monenlaisia vaihdellen fyysisen vahingon aikaansaamisesta tai yksityisten ja yksilöivien tietojen varastamisesta taloudellisen hyödyn tavoitteluun (Felt, Finifter ym., 2011, ss. 3-4; Sikorski ja Honig, 2012, s. xxviii). Yhteistä haittaohjelmille on kuitenkin monesti se, että niiden laatijat tai levittäjät tavoittelevat taloudellista hyötyä. Esimerkiksi Suarez-Tangil ym. (Suarez-Tangil, J. Tapiador ym., 2014, s. 963) kirjoittavat haittaohjelmien motiivien olevan nykyään pitkälti taloudellisia. Samansuuntaisia ajatuksia ovat tuoneet esille myös Viestintäviraston Kyberturvallisuuskeskuksen Juha Tretjakov sekä F-Securen Mikko Hyppönen (Rissanen ja Koivuranta, 2016).

2.1 Haittaohjelmien leviäminen

Haittaohjelmia levitetään eri tavoin. Levittämisessä hyödynnetään useasti ohjelmistojen haavoittuvuuksia sekä *käyttäjän manipulointia* (social engineering) tavalla tai toisella. (Kaspersky, 2009; Viestintävirasto, 2015b) Tarkastellaan seuraavaksi joitakin yleisiä haittaohjelmien leviämis- ja levitystapoja, joihin liittyvät edellä mainitut seikat.

2.1.1 Ohiajolataus

Haittaohjelmia levitetään usein *ohiajolataukseksi* (drive-by download) kutsutun tekniikan avulla (Viestintävirasto, 2015b). Ohiajolatauksessa haittaohjelma ladataan web-sivustolta laitteelle käyttäjän tietämättä. Käyttäjän laite voi siis tulla saastutetuksi vain vierailemalla web-sivustolla; mitään muuta tähän ei tarvita.

Yleensä ohiajolatausten automatisoinnissa hyödynnetään *haavoittuvuuksien hyväksikäyttötyökalua* (exploit kit). (Kaspersky, 2009; Microsoft, 2011)

Haavoittuvuuksien hyväksikäyttötyökalu määrittää selaimen ja käyttöjärjestelmän kokoonpanon HTTP-pyynnön tiedoista. Näistä kokoonpanotiedoista määritetään, mitä kohdekoneen haavoittuvuutta käytetään hyväksi. (Kaspersky, 2009) Kun kohdekoneen haavoittuvuus tai haavoittuvuudet on määritetty, oikea haavoittuvuuden hyväksikäyttö tarjotaan kohdekoneelle. Tällöin haavoittuvuutta hyväksikäyttävä ohjelmakoodi lataa haittaohjelman ja kone tulee saastutetuksi. (Kaspersky, 2009; Viestintävirasto, 2015a)

2.1.2 Troijan hevonen

Haittaohjelmia levitetään hyvin usein niin sanottuna *Troijan hevosena* (F-Secure, 2015b, s. 10, katso kuvio "Malware by type"). Nimitys "Troijan hevonen" (tai lyhyesti myös "troijalainen") viittaa kreikkalaisen myytin puiseen Troijan hevoseen (F-Secure, 2016[f]), jonka kätköihin piiloutuneet kreikkalaiset sotilaat sittemmin nousivat ja valloittivat Troijan kaupungin yhdessä muun kreikkalaisen armeijan kanssa (contributors, 2016[c]).

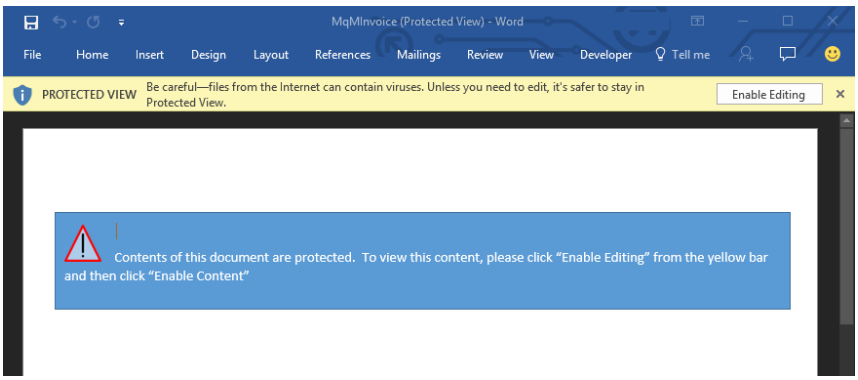
Myytin esikuvan mukaisesti troijalainen on esimerkiksi sovellus, joka on tarkoituksella rakennettu näyttämään tavalliselta ja hyödylliseltä, mutta johon on kuitenkin kätketty haitallinen toiminnollisuus. Tällöin käyttäjä asentaa sovelluksen huomaamatta sen olevan haitallinen – samoin kuin troijalaiset vetivät kreikkalaisia sotilaita sisältävän puisen hevosen kaupunkiinsa osaamatta aavistaa hevosen todellista luonnetta. Troijan hevonen voi olla myös esimerkiksi asiakirja tai mediatiedosto, kuten video tai kuva. (F-Secure, 2016[f]; contributors, 2016[c])

Troijalaiset näyttäisivät olevan myös suurin Androidille kohdennettu haittaohjelmaryhmä. F-Securen vuoden 2015 uhkaraportissa (F-Secure, 2015b, s. 14) kahdeksan kymmenestä yleisimmästä Android-haittaohjelmasta kuului nimenomaan tähän ryhmään. Troijalaisena voidaan levittää muita erityyppisiä haittaohjelmia (contributors, 2017a), kuten kiristysohjelmia (contributors, 2017[d]), joihin palataan hieman myöhemmin.

2.1.3 Sähköposti

Viimeisenä tapana tarkastellaan haittaohjelmien levittämistä sähköpostitse. Viestintäviraston Kyberturvallisuuskeskuksen vuosiraportin (2015) mukaan sähköpostiviestit ovat edelleen yleinen tapa levittää haittaohjelmia. Haittaohjelmat leviävät joko suoraan sähköpostin liitetiedostona tai sähköpostitse lähetettävien haitallisten linkkien kautta. (Viestintävirasto, 2015c)

Microsoftin TechNet-blogissa on kuvattu esimerkin avulla tilanne, jossa yrityksen työntekijälle on tarkoituksella lähetetty haitallinen Microsoft Word -dokumentti sähköpostin liitteenä. Word avaa dokumentin *suojatussa näkymässä* (protected view), jossa makrot on poistettu käytöstä. Käyttäjän on kuitenkin mahdollista ottaa makrot käyttöön napsauttamalla kuvassa 2.1 näkyvää "Enable editing" -painiketta. (Microsoft, 2017; Microsoft, ei julkaisupäivää)



Kuva 2.1: Microsoft Word -ohjelmiston "Enable editing" -painike, jota napsauttamalla makroille annetaan suoritusoikeudet. (Microsoft, 2017).

2.2 Haittaohjelmien torjunta

2.2.1 Torjuntaohjelmistot

Haittaohjelmien torjuntaohjelmistoja on ollut jo pitkään saatavilla henkilökohtaisille tietokoneille. Nämä ohjelmistot ovat kehittyneet ensimmäisistä, yksinkertaisista digitaalisiin allekirjoituksiin perustuvista järjestelmistä huomattavasti monimutkaisempiin ja laajempiin järjestelmiin (F-Secure, 2016[g]). Haittaohjelmien torjuntaohjelmistoja on kehitetty myös mobiililaitteille, kuten esimerkiksi Androidille (F-Secure, 2017).

Nykyisissä haittaohjelmien torjuntaohjelmistoissa yhdistyy allekirjoituksella tapahtuva tunnistus muun muassa tiedoston tunnettuuden arviointiin sekä sovelluksen käyttäytymiseen. Käyttäytymistä voidaan tarkkailla suorittamalla sovellus ensin virtuaalisessa *hiekkalaatikossa* (sandbox). (F-Secure, 2016b, ss. 3,5-6; F-Secure, 2016[g])

Virtuaalisella hiekkalaatikolla tarkoitetaan tässä asiayhteydessä sitä, että esimerkiksi internetistä ladattu sovellus suoritetaan virtualisoidussa ympäristössä, jotta sen toimintaa voidaan tarkastella. Tällä tavoin voidaan havaita, käyttäytyykö sovellus haitallisella tavalla vai ei. Kun sovelluksen käyttäytymismalli yhdistetään tunnistusprosessiin, voidaan havaita ja tunnistaa myös entuudestaan tuntemattomia haittaohjelmia (F-Secure, 2016b, ss. 5-7).

2.2.2 Hyökkäyspinnan pienentäminen

Haittaohjelma-tartunnan riskiä voidaan pienentää pienentämällä hyökkäyspintaa (Viestintävirasto, 2015b). Esimerkiksi aiemmin jo lähes hävinneet ja nyttemmin jälleen yleistyneet makrovirukset toimivat siten, että ne pyrkivät huijaamaan käyttäjää aktivoimaan – esimerkiksi Microsoft Word -ohjelmiston – makrotuen (Viestintävirasto, 2015c, s. 9; Kaspersky, 2016[c]). Kun käyttäjä aktivoi tuen, saa haittaohjelma esimerkiksi luku- ja kirjoitusoikeudet käyttäjän tiedostoihin (Sophos, 2016). Microsoft on kuitenkin lisännyt Office-pakettiin ominaisuuden, jonka avulla yrityksessä voidaan *ryhmäkäytänteen* (group policy) avulla estää verkosta ladattujen dokumenttien sisältämien makrojen suoritus (Microsoft, 2017). Tällöin hyökkäyspinta pienenee, makrovirukset eivät toimi ja haittaohjelmatartunnan riski vähenee. (Dormann, 2016)

2.2.3 Kouluttaminen ja tiedottaminen

Osa haittaohjelmien torjuntaa on myös käyttäjien kouluttaminen. F-Securen Hyppönen kertoo, että on olemassa kahdenlaisia ongelmia: teknisiä ongelmia ja ihmisiin liittyviä ongelmia. Ohjelmistojen ongelmat voidaan korjata päivityksellä, mutta ihmisten kohdalla ongelma on vaikeampi: asioita täytyy toistaa uudestaan ja uudestaan. (W. Wei, 2016)

Kouluttamiseen ja tiedottamiseen liittyy myös se, että älylaitteiden käyttäjät etsivät useasti kolmannen osapuolten sovelluskaupoista ilmaiseksi sovelluksia, jotka ovat maksullisia esimerkiksi virallisessa Google Play -sovelluskaupassa. Tällaiset – mahdollisesti laittomatkin – sovelluskaupat ovat kuitenkin “hyvä” alusta haittaohjelmien leviämiseen, kuten Suarez-Tangil ym. toteavatkin. (Suarez-Tangil, J. Tapiador ym., 2014, s. 963)

2.2.4 Ohjelmistojen ajantasaisuus

Ohjelmistojen päivittäminen on tärkeässä roolissa haittaohjelmien torjunnassa (Viestintävirasto, 2015b). Ohjelmistoissa havaitaan aika ajoin vakaviakin puutteita ja haavoittuvuuksia. Tällöin järjestelmän tai ohjelmiston valmistaja mahdollisesti korjaa haavoittuvuuden ja julkaisee päivityksen.

Androidin kohdalla tämä ei – laitteesta riippuen – toimi, sillä laitteiden valmistajat eivät välttämättä julkaise päivityksiä halvimpiin laitemalleihinsa. Lisäksi varsinaiseen käyttöjärjestelmään kohdistuvat päivitykset joutuvat tekemään pitkän matkan Googlelta käyttäjän puhelimeen. (F-Secure, 2015a)

Esimerkiksi vuonna 2015 julkistettu Stagefright-haavoittuvuuksien perhe koski noin miljardia Android-laitetta (contributors, 2016[b], katso otsikon “History”

alta). Haavoittuvuutta on kuitenkin erittäin vaikea korjata yllämainituista syistä ja todennäköisesti suurin osa tästä haavoittuvuudesta kärsineistä laitteista on edelleen haavoittuvia (F-Secure, 2015a).

2.3 Haittaohjelmaesimerkkejä

Tarkastellaan seuraavaksi haittaohjelmia kahden esimerkin kautta. Toinen esimerkeistä, DroidDream, on Android-haittaohjelma (F-Secure, 2016[e]) ja toinen, CryptoWall, puolestaan saastuttaa Windows-laitteita (Cyber Threat Alliance, 2015; F-Secure, 2015b).

2.3.1 DroidDream-troijalainen

DroidDream on Android-haittaohjelma, jonka ominaisuuksiin kuuluu muun muassa käyttäjän ja laitteen yksilöivien tietojen varastaminen. Näihin tietoihin kuuluvat laitteen mallitunniste ja `IMEI`-tunnus, käytössä oleva kieli sekä operaattorin `IMSI`-tunniste. Lisäksi se voi myös hankkia käyttöjärjestelmän pääkäyttäjän (root) oikeudet, joiden avulla sen on mahdollista tehdä käytännössä mitä vain, kuten esimerkiksi asentaa muita sovelluksia käyttäjän tietämättä. (F-Secure, 2016[e])

DroidDream lähettää varastamansa tiedot etäpalvelimelle (remote server). Kuten edellä todettiin, `IMEI`-tunnus on eräs DroidDreamin varastamista tiedoista. (F-Secure, 2016[e]) `IMEI` on mobiililaitteen 15 merkkiä pitkä yksilöllinen tunniste, jonka alkuperäinen tarkoitus oli estää varastettujen laitteiden pääsy `GSM`-verkkoon: `IMEI`-tunnuksen perusteella lailliset laitteet päästetään verkkoon, mutta varastetuilta ja siten mustalle listalle päätyneiltä laitteilta estetään pääsy matkapuhelinverkkoon. (Grzonkowski ym., 2014, s. 41; Felt, Finifter ym., 2011, s. 5)

Varastettuja `IMEI`-tunnuksia käytetään mahdollisesti myös varastettujen laitteiden vastaavien tunnusten korvaamiseen. Jos varastetun laitteen tunnus lisätään mustalle listalle, laite ei voi enää yhdistää verkkoon. Tämän vuoksi varastetun laitteen `IMEI`-tunnus korvataan varastetulla, mutta edelleen toimivalla `IMEI`-tunnuksella. (Felt, Finifter ym., 2011)

Kuten Felt ym. (Felt, Finifter ym., 2011) toteavat, varmuutta varastettujen tietojen käyttötarkoituksesta ei ole. Kirjoittaja huomasi tutkielmaa tehdessään, että on hankala löytää kelvollisia lähteitä, joista kävisi ilmi, mitä varastetuilla `IMEI`-tunnuksilla itseasiassa tehdään. Kirjoittaja lähestyikin Suomen Keskusrikospoliisia asiassa, mutta tätä kirjoittaessa vastausta ei ole saatu. Laajemman alueen, kuten Euroopan tai Yhdysvaltain tasolla informaatio olisi toki kattavampi ja käyttökelpoisempi, mutta toistaiseksi kirjoittaja ei ole ottanut yhteyttä esimerkiksi Europolin tai FBI:n kaltaisiin tahoihin.

2.3.2 CryptoWall-kiristysohjelma

Kiristysohjelmia levitetään troijalaisina (contributors, 2017[d]). Kiristysohjelma on haittaohjelma, joka salaa käyttäjän laitteella olevat tiedostot ja vaatii maksua salauksen purkamiseen tarvittavan avaimen vastineeksi. Mikäli käyttäjä ei maksa lunnaita, seurauksena voi olla tiedostojen pysyvä menetys. (Trend Micro, 2017, katso kohdasta “If I get infected, how to remove the ransomware?”; Kaspersky, 2016[b]) Kuvassa 2.2 on erään kiristysohjelman lukitusruutu, jossa kerrotaan käyttäjälle tilanteesta ja ohjeistetaan, miten saada tiedot takaisin.

Kuva 2.2: Kuvankaappaus erään kiristysohjelman lukitusruudusta. Ruudussa kerrotaan, että käyttäjän tiedot on salattu ja ainoa keino saada ne takaisin, on maksaa kiristäjille. (F-Secure, ei julkaisupäivää[a])

Kiristysohjelmien levittäjille ei kuitenkaan ole suositeltavaa maksaa lunnaita, sillä juuri lunnaiden maksu tekee kiristysohjelmien levittämisestä kukoistavaa “liiketoimintaa” (Kaspersky, 2016[b], katso usein kysytyt kysymykset: "Why not just pay the ransom?"). Lisäksi Kasperskyn tutkimuksen mukaan jopa 20 % niistä, jotka maksoivat vaaditut lunnaat, eivät saaneet tiedostojaan takaisin (Kaspersky, 2017[b]).

Käytettävästä laitteistosta erillään pidettävä ja riittävän usein päivitettävä varmuuskopio omista henkilökohtaisista tiedostoista on hyvä keino suojautua kiristysohjelmia vastaan (Ducklin, 2013). Esimerkiksi Trend Micro suosittelee käyttämään niin sanottua 3-2-1-varmuuskopiointisääntöä (Trend Micro, 2017). Sääntö sanoo muun muassa, että tulisi tehdä 3 varmuuskopiota, joista yksi sijaitsee fyysisesti toisessa sijainnissa. Tällöin esimerkiksi tulipalon sattuessa yksi varmuuskopio on suojassa. (Trend Micro, 2013)

Joidenkin kiristysohjelmien salaamia tiedostoja varten on olemassa purkuoh-

jelma, joka kykenee purkamaan salauksen ilman, että käyttäjän tarvitsee maksaa kiristäjälle. (Kaspersky, 2016[b]) Muita suojautumiskeinoja tarkastellaan myöhemmin hieman enemmän.

Hyppösen (Rissanen ja Koivuranta, 2016) mukaan kiristysohjelmista on nykypäivänä tullut yksi suurimmista ongelmista. Kiristysohjelmat ovatkin varsin ikävä yllätys esimerkiksi yrityksen tietoverkossa, jossa saastuneeseen laitteeseen on liitetty verkkolevyasemia. Tällöin kiristysohjelma saattaa salata myös verkkolevyasemalla olevat tiedostot. (Ducklin, 2013)

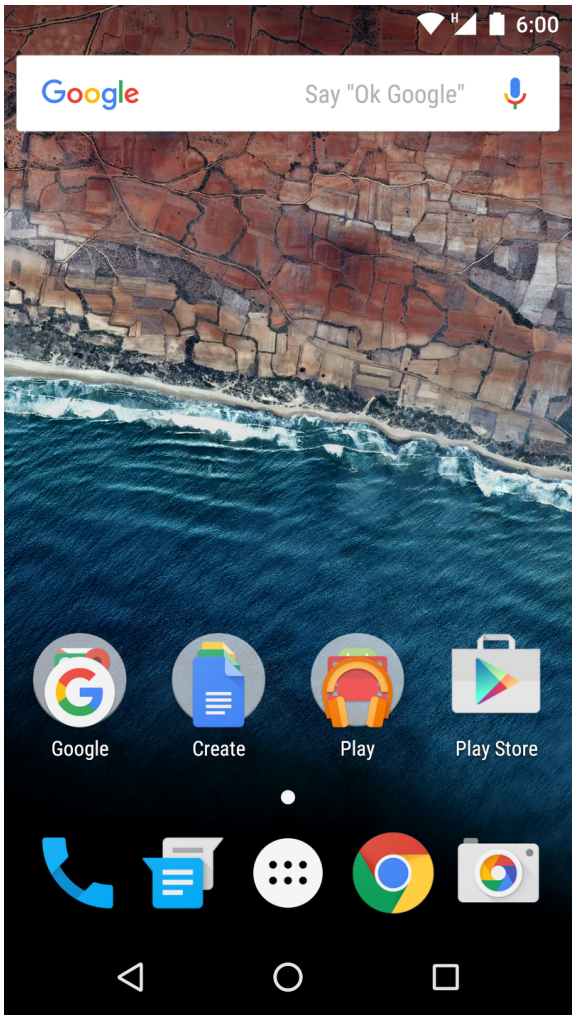
Lisäksi kiristysohjelmat voivat aiheuttaa vakavia ongelmia esimerkiksi terveydenhuollon järjestelmissä (Rissanen ja Koivuranta, 2016). Tämän tyyppisiä haittaohjelmia onkin jo löydetty sairaaloista muun muassa Yhdysvalloissa (NBC, 2016), Saksassa ja Suomessa (Rissanen ja Koivuranta, 2016).

CryptoWall 3 on eräs esimerkki kiristysohjelmasta. Cyber Threat Alliancen tutkimuksesta käy ilmi, että sen aikaansaamat vahingot nousivat jopa yli 300 miljoonaan dollariin. (Cyber Threat Alliance, 2015, s. 4-5) CryptoWall saastuttaa PC-tietokoneita, mutta vastaavia haittaohjelmia on tehty myös Androidille (Cyber Threat Alliance, 2015; F-Secure, 2015b). Eräs esimerkki Android-kiristysohjelmasta on SLocker, joka löytyy edellä jo useasti mainitusta F-Securen vuoden 2015 uhkaraportista toiselta sijalta. Samasta raportista käy myös ilmi, että SLocker luokitellaan troijalaiseksi. (F-Secure, 2015b)

3 Android

Android on suosittu Linux-ytimeen perustuva mobiilikäyttöjärjestelmä, jota kehittää pääasiallisesti Google. Sen ensimmäinen kaupallinen versio, versio 1.0, julkaistiin vuonna 2008. (contributors, 2016[a]; Drake ym., 2014, s. 2) Android on siis tätä kirjoitettaessa jo lähes vuosikymmenen ikäinen ohjelmistoprojekti[1]. Näiden vuosien aikana Android on kehittynyt paljon eri osa-alueilla, joista yksi tärkeimmistä on turvallisuus (Elenkov, 2014, s. xxi). Androidin turvallisuuskysymykset ovatkin oleellisia tutkielman kannalta ja niitä tarkastellaan myöhemmin.

Kuvassa 3.1 on ruudunkaappaus Androidin käyttöliittymästä. Kuvan alalaidassa nähdään muun muassa Google Play -sovelluskaupan kuvake, jonka kautta käyttäjä voi asentaa laitteeseensa sovelluksia.



Kuva 3.1: Ruudunkaappaus Android 6 -käyttöjärjestelmästä (Google, 2015).

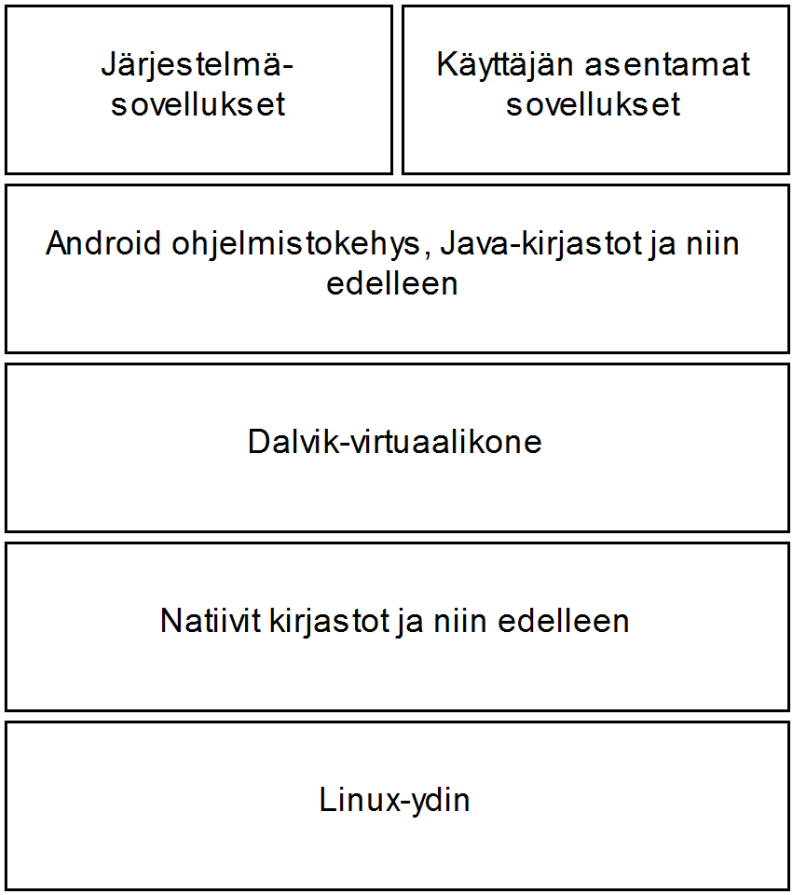
Uusin Android-versio tutkielmaa kirjoitettaessa on versio 7.1 (contributors, 2016[a]). Tätä edeltävässä versiossa eli versiossa 6, Androidin oikeusjärjestelmään

[1]Jatkossa, kun Androidin tiettyihin toiminnallisuuksiin tai ominaisuuksiin viitataan, tarkoitetaan lähteen tarkoittamaa Android-versiota. Kehitystyön edetessä jotkin asiat voivat muuttua eivätkä välttämättä ole tosia kaikkien versioiden kohdalla.

tehtiin muutoksia (contributors, 2016[a]) – josta myöhemmin lisää – ja muun muassa näiden muutosten vuoksi tutkielmassa on huomioitu vain versiota 6 aiemmat Android-versiot. Tarkastellaan seuraavaksi tutkielman kannalta oleellisia, Androidin arkkitehtuuriin, sovelluksiin ja turvallisuuteen ja liittyviä seikkoja.

3.1 Arkkitehtuuri ja sovellukset

Android rakentuu useasta eri kerroksesta (Drake ym., 2014, ss. 25-26; Elenkov, 2014, s. 2). Kuvassa 3.2 on yksinkertaistettu ja mukautettu versio Elenkovin esittämästä Androidin kerroksellisesta arkkitehtuurista, jossa alimmalla kerroksella sijaitsee Linux-ydin. Ytimen päälle rakentuvat muut kerrokset, kuten Dalvik-virtuaalikone, jossa Android-sovellukset pääasiallisesti suoritetaan (Elenkov, 2014, s. 2).



Kuva 3.2: Yksinkertaistettu ja mukautettu versio Elenkovin (Elenkov, 2014, s. 2) esittämästä Androidin kerroksellisesta arkkitehtuurista, jossa Linux-ydin on alimmalla ja sovellukset ylimmällä kerroksella.

Tutkielman kannalta oleellinen osa Androidia ovat *sovellukset* (applications), jotka nähdään kuvassa 3.2 ylimmällä kerroksella. Sovellukset jakautuvat kahteen osaan: Androidin järjestelmäsovelluksiin sekä käyttäjän itse asentamiin sovelluk-

siin (Elenkov, 2014, s. 10). Suurin osa sovelluksista on kuitenkin loppukäyttäjän laitteeseensa asentamia.

Loppukäyttäjä voi asentaa sovelluksia muutamallakin eri tavalla. Suurin osa käyttäjistä asentaa kuitenkin sovelluksensa sovelluskaupoista, joista Google Play lienee tunnetuin. (Elenkov, 2014, s. 61) Lisäksi sovelluksia voi asentaa myös tietokoneen kautta joko `adb`-työkalun avulla tai siirtämällä sovelluspaketin ensin tietokoneelta kohdelaitteelle ja käynnistämällä asennuksen manuaalisesti tiedostoselaimen kautta. Jotkut asennusmenetelmät soveltuvat lähinnä sovelluskehittäjille. (Elenkov, 2014, s. 61; contributors, 2017[b]; contributors, 2017[e])

Sovellukset ohjelmoidaan pääasiallisesti Javalla (contributors, 2016[a]). Androidin versioon 4.4. asti sovellukset suoritettiin pääasiallisesti Dalvik-virtuaalikoneessa, mutta versiosta 5 lähtien Dalvikin korvasi *Androidin ajonaikaisympäristö* (Android Runtime, ART) kokonaan. (contributors, 2017[c])

Androidin ajonaikaisympäristö käyttää Dalvik-tavukoodia syötteenään, koska se takaa taaksepäin yhteensopivuuden vanhempien, Dalvikia käyttävien Android-laitteiden kanssa. Ajonaikaisympäristössä ei kuitenkaan suoriteta Dalvik-tavukoodia, vaan se käännetään asennuksen yhteydessä natiiviksi konekoodiksi, joka puolestaan suoritetaan ART-ympäristössä. (contributors, 2017[c]) Siten sovellusten, jotka on alun perin ohjelmoitu Dalvikille, pitäisi toimia myös ART-ympäristössä (Google, 2017a). Koska sama sovellus toimii sekä Dalvik-virtuaalikoneessa että Androidin ajonaikaisympäristössä, näitä ei ole tarvetta käsitellä erikseen tässä tutkielmassa.

Android-sovelluspaketti eli *APK* (Android application package) on käytännössä ZIP-pakattu tiedosto. Tällöin käyttäjä voi halutessaan purkaa sen helposti ja tarkastella sen sisältöä (Elenkov, 2014, s. 52). Kun Javalla ohjelmoitu sovellus on käännetty Dalvik-tavukoodiksi, paketoidaan se APK-tiedostoksi yhdessä esimerkiksi sovelluksen käyttämien mediatiedostojen kanssa. Nyt valmis sovelluspaketti voidaan julkaista esimerkiksi Google Play -sovelluskaupassa, josta loppukäyttäjä voi asentaa sen laitteeseensa. (Moonsamy ym., 2014, s. 123)

Jokainen APK-sovelluspaketti sisältää myös `AndroidManifest.xml`-tiedoston. Kyseinen manifest-tiedosto on pakollinen osa sovellusta ja se sisältää paljon erilaista informaatiota muun muassa sovelluksen komponenteista. Tässä tiedostossa määritellään myös sovelluksen vaatimat oikeudet. (Drake ym., 2014, s. 35)

Listauksessa 1 on katkelma k9mail-sähköpostisovelluksen[2] `AndroidManifest`-tiedostoa. Katkelma sisältää muutaman esimerkin sovelluksen oikeusvaatimuksista: listauksesta nähdään, kuinka sovellus vaatii esimerkiksi `INTERNET`-oikeutta, jotta se voi muodostaa verkkoyhteyden sekä `READ_CONTACTS` -oikeuden, jotta se voi hyödyntää laitteeseen tallennettuja yhteystietoja.

[2]https://github.com/k9mail/k-9

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.fsck.k9">

    <uses-permission
        android:name="android.permission.READ_CONTACTS"/>
    <uses-permission
        android:name="android.permission.INTERNET"/>
    <uses-permission
        android:name="android.permission.WAKE_LOCK"/>

</manifest>
```

Listaus 1: Esimerkki oikeuksien vaatimisesta `AndroidManifest`-tiedostossa. Listaus ei sisällä kaikkia sovelluksen vaatimia oikeuksia, vaan listausta on lyhennetty ja rivitetty lukemisen helpottamiseksi. Esimerkki on k9mail sähköpostisovelluksesta.

Vaaditut oikeudet myönnetään sovellusta asennettaessa. Mikäli käyttäjä ei halua myöntää sovellukselle sen vaatimia oikeuksia, jää ainoaksi vaihtoehdoksi asennuksen peruuttaminen. Mikäli käyttäjä haluaa myöhemmin peruuttaa myöntämänsä oikeudet, on hänen poistettava sovellus laitteesta. (Elenkov, 2014, ss. 23-25) Versiosta 6 eteenpäin näin ei kuitenkaan enää ole, vaan oikeuksia voidaan sekä myöntää että kieltää asennuksen jälkeenkin. (Google, 2017[b]) Oikeuksiin perehdytään hieman tarkemmin seuraavassa aliluvussa Androidin turvallisuuskysymysten yhteydessä.

3.2 Turvallisuuskysymykset

Kun sovellus asennetaan laitteeseen, annetaan sille oma käyttäjätunnus sekä data-hakemisto. Sovellus suoritetaan omassa prosessissaan, sille annetun käyttäjätunnuksen oikeuksin, eikä muilla sovelluksilla ole pääsyä prosessin muistiin. Myös sovelluksen data-hakemistoon on pääsy vain data-hakemiston omistavalla sovelluksella; muilla sovelluksilla kyseiseen hakemistoon ei ole pääsyä. Edellä kuvatut toimenpiteet eristävät sovellukset toisistaan. Näitä eristystoimenpiteitä kut-

sutaan *hiekkalaatikoinniksi*[3] (sandboxing). (Elenkov, 2014, ss. 5, 11-13)

Hiekkalaatikoidulla sovelluksella ei ole juuri mitään oikeuksia hiekkalaatikkonsa ulkopuolelle. Se ei siis voi käyttää esimerkiksi laitteeseen talletettuja yhteystietoja. (Elenkov, 2014, ss. 14, 47) Tällöin esimerkiksi sijaintiin perustuvia muistutuksia antava sovellus ei toimisi, koska sovellus tarvitsee toimiakseen oikeuden käyttää laitteen sijaintia (esimerkiksi GPS). Oikeutta käyttää laitteen tai käyttöjärjestelmän resursseja, tietoja ja palveluita, kuten GPS-paikannusta tai laitteeseen tallennettuja yhteystietoja, kontrolloidaan oikeusjärjestelmän kautta (Elenkov, 2014, s. 14).

Android-järjestelmä sisältää `AndroidManifest.xml`-tiedoston, jossa se määrittelee joukon *järjestelmäoikeuksia* (system permissions) (Elenkov, 2014, s. 37). Järjestelmäoikeuksille on määritelty *suojaustasot* (protection level), jotka määrittelevät, kenelle ja mitä oikeuksia voidaan myöntää. Suojaustaso vaikuttaa myös siihen, myönnetäänkö oikeudet automaattisesti vai tarvitaanko myöntämiseen käyttäjän suostumus. Suojaustasoja on neljä: *normaali* (normal), *vaarallinen* (dangerous), *allekirjoitus* (signature) sekä *järjestelmä tai allekirjoitus* (system or signature). (Elenkov, 2014, pp. 24-26)

Normaali suojaustaso

Normaalin suojaustason oikeuksille ei vaadita sovelluksen asennusvaiheessa erillistä käyttäjän suostumusta, vaan ne myönnetään automaattisesti (Elenkov, 2014, ss. 24-25). Esimerkiksi `SET_WALLPAPER`[4]-oikeus on normaalin suojaustason oikeus, jota tarvitaan laitteen taustakuvan vaihtamiseen.

Vaarallinen suojaustaso

Vaarallisen suojaustason oikeuksille taas vaaditaan käyttäjän suostumus sovelluksen asennuksen yhteydessä. Sovelluksen asentaja voi joko hyväksyä oikeusvaatimukset tai hylätä vaatimukset, jolloin ainoaksi vaihtoehdoksi jää asennuksen keskeyttäminen. (Elenkov, 2014, s. 25) Esimerkki vaarallisen suojaustason oikeuksista on `SEND_SMS`[5], jota tarvitaan `SMS`-viestien lähettämiseen. Sovellus, jolle on myönnetty tämä kyseinen oikeus, voi siis lähettää `SMS`-viestejä.

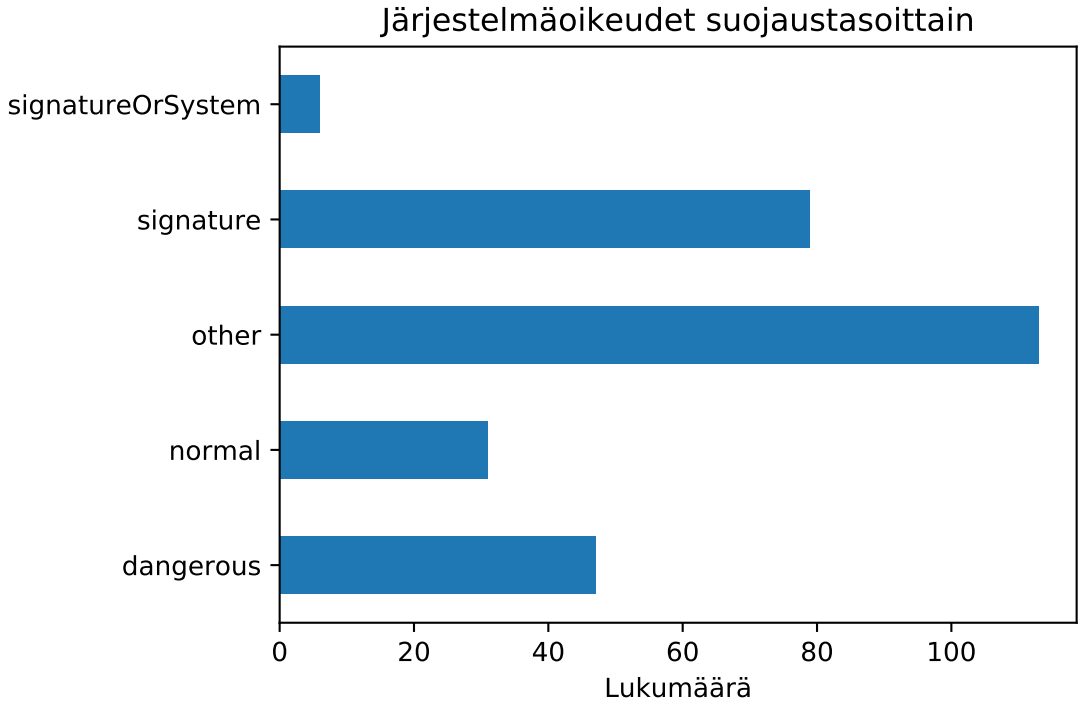
Allekirjoitus ja järjestelmä tai allekirjoitus suojaustasot

Allekirjoitus-suojaustason oikeuksia voidaan myöntää vain sellaisille sovelluksille, jotka ovat allekirjoitetut samalla digitaalisella allekirjoituksella kuin oikeuden määritellyt taho (Elenkov, 2014, ss. 26, 39). Allekirjoituksella viitataan tässä kryptografiseen allekirjoitukseen, jolla voidaan varmistaa muun muassa se, että so-

[3]Tässä asiayhteydessä hiekkalaatikointi on eri kuin mitä on tarkasteltu haittaohjelmien torjuntaohjelmistojen yhteydessä luvussa 2.2. Karkealla tasolla molemmissa on kyse samasta asiasta: rajoitetusta toimintaympäristöstä muun muassa turvallisuuskysymysten vuoksi.

[4]https://developer.android.com/reference/android/Manifest.permission.html

[5]Katso alaviite 4



Kuva 3.3: Androidin järjestelmäoikeuksien lukumäärät suojaustasoittain. Muut (other) -kategoriaan kuuluvat muun muassa suojauslipulla vahvennetut oikeudet. Lisätietoja nämä oikeudet määritelleestä `AndroidManifest`-tiedostosta on kappaleessa 4.

velluksen päivitykset tulevat vain auktorisoidulta taholta (Drake ym., 2014, s. 38; Elenkov, 2014, s. 24). Kun kysymyksessä on Androidin järjestelmäoikeudet, tarvitaan esimerkiksi laitevalmistajan digitaalinen allekirjoitus, jotta oikeus voidaan myöntää. (Elenkov, 2014, s. 39; Felt, Chin ym., 2011, s. 628)

Järjestelmä tai allekirjoitus -suojaustason oikeus eroaa edellisestä siten, että oikeus voidaan myöntää myös sellaiselle sovellukselle, joka on asennettu tiettyyn suojattuun järjestelmäosioon, vaikka sovellusta ei olisikaan allekirjoitettu oikeuden määrittäneen tahon allekirjoituksella (Elenkov, 2014, s. 26).

Kuvassa 3.3 on Androidin järjestelmäoikeuksien lukumäärät suojaustasoittain. Edellä kuvattujen neljän suojaustason lisäksi kuvassa on muut (other)-kategoria, johon on sisällytetty *suojauslipulla* (protection flag) vahvennetut oikeudet (Elenkov, 2014, s. 38) sekä ilman suojaustasoa olevat oikeudet, joita rajaamiseen käytetyssä manifest-tiedostossa oli vain kaksi.

Sovelluksen ei kuitenkaan ole välttämätöntä käyttää kaikkia vaatimiaan oikeuksia. Moonsamy ym. kutsuivatkin pyydetyiksi tai *vaadituiksi oikeuksiksi* (required permissions) sovelluksen manifest-tiedostossaan vaatimia oikeuksia sekä *käytetyiksi oikeuksiksi* (used permissions) niitä oikeuksia, joita sovellus varsinaisesti käyttää. (Moonsamy ym., 2014, ss. 123, 131) Tässä tutkielmassa käytettiin muut-

tujina vain vaadittuja oikeuksia. Käytössä ollutta datajoukkoa ja muuttujia tarkastellaan paremmin seuraavassa luvussa.

4 Data ja muuttujat

Datajoukko koostui hieman yli 120 000 tapauksesta ja 276 muuttujasta. Tapaukset ovat Android-sovelluksia, jotka jakautuvat kahteen luokkaan: hyvänlaatuisiin ja haitallisiin. Hyvänlaatuisia sovelluksia, kuten myös haitallisia, on noin puolet koko datajoukosta. Kaikki tapaukset on kerätty vuosien 2010 ja 2014 välisenä aikana: hyvänlaatuiset ovat vuosilta 2010—2014 ja haitalliset vuoden 2014 ensimmäiseltä puoliskolta (F-Secure, ei julkaisupäivää[b]).

Muuttujia ovat sovellusten vaatimat Androidin järjestelmäoikeudet. Yksittäinen sovellus voi vaatia Androidin järjestelmäoikeuksien lisäksi myös muiden sovellusten määrittelemiä oikeuksia (Elenkov, 2014, ss. 42-43). Tutkielman kannalta oleellisia olivat kuitenkin vain Androidin määrittelemät järjestelmäoikeudet.

Oikeudet rajattiin kattamaan vain Androidin järjestelmäoikeudet. Rajaamiseen käytettiin `AndroidManifest`[6]-tiedostoa, jossa nämä järjestelmäoikeudet määritellään. Käytetty manifest-tiedosto valittiin Androidin viimeisten versioiden joukosta ennen versiota 6.

Taulukosta 1 nähdään vaadittujen järjestelmäoikeuksien määrät. Hyvänlaatuiset sovellukset pyytävät keskimäärin noin 10 oikeutta sovellusta kohden, kun taas haitalliset keskimäärin 13 erilaista oikeutta sovellusta kohden. Keskihajonta hyvänlaatuisten sovellusten oikeusvaatimuksissa oli 9 ja haitallisten vastaavasti 7. Kokonaismäärällisesti haitalliset sovellukset pyytävät huomattavasti enemmän oikeuksia kuin hyvänlaatuiset sovellukset.

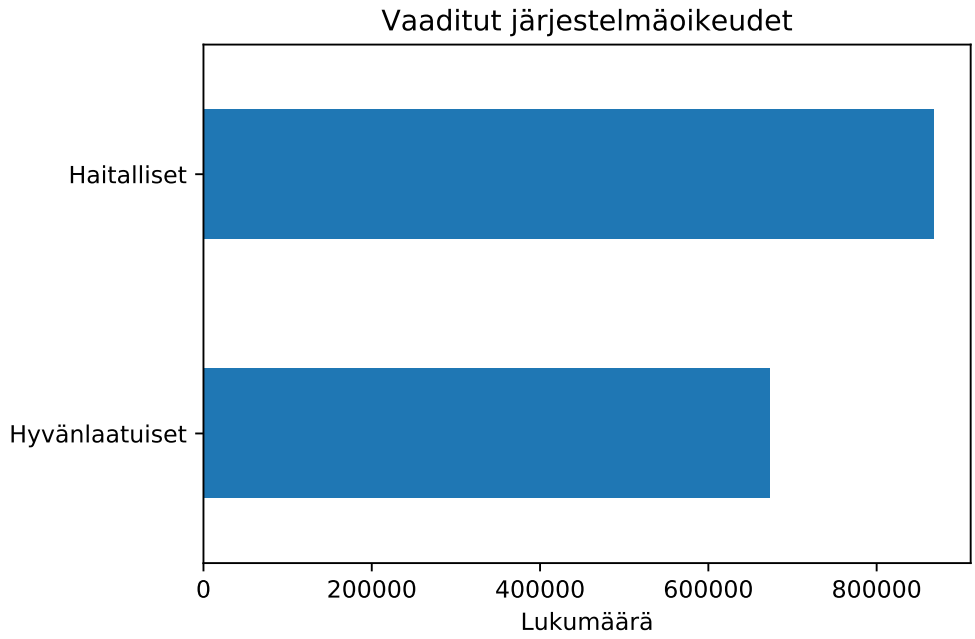
Luokka	Lukumäärä	Keskiarvo	Keskihajonta
Hyvänlaatuinen	672 763	10	9
Haitallinen	868 321	13	7

Taulukko 1: Vaadittujen oikeuksien määrät koko datajoukossa. Keskiarvot ja keskihajonnat on pyöristetty lähimpään kokonaislukuun.

Kuvassa 4.1 on pylväskaavio kaikista vaadituista oikeuksista. Koko datajoukossa esitettiin yhteensä hieman reilut 1,5 miljoonaa oikeusvaatimusta.

Taulukossa 2 on kymmenen haitallisten sovellusten eniten vaatimaa oikeutta sekä hyvänlaatuisten sovellusten vastaavat lukumäärät. Taulukosta nähdään, että haitalliset sovellukset vaativat lähes kaikkia (listattuja) oikeuksia enemmän kuin hyvänlaatuiset. Hyvänlaatuiset sovellukset pyysivät haitallisia sovelluksia enemmän vain yhtä oikeutta, joka on taulukon toiseksi viimeisellä rivillä sijaitseva `WAKE_LOCK`-oikeus. Tätä oikeutta käytetään muun muassa näytön päällä olemisen säätelemiseen (Felt, Ha ym., 2012, s. 7).

[6]Tiedoston Git SHA-1-hajautusarvo on c2640ef66fe5cc87ef3978592e68ed078a08f6e2



Kuva 4.1: Pylväskaavio vaadituista oikeuksista. Yhteensä oikeusvaatimuksia koko datajoukossa oli hieman reilut 1,5 miljoonaa.

Kolmansien osapuolien ohjelmistokehittäjät voivat käyttää vain normaalin tai vaarallisen tason järjestelmäoikeuksia. Tämä johtuu muun muassa siitä, että esimerkiksi digitaaliseen allekirjoitukseen perustuvia järjestelmäoikeuksia ei myönnetä kolmansille osapuolille, vaan niitä käyttävät ainoastaan esimerkiksi laitevalmistajat. (X. Wei ym., 2012, s. 33)

Käytettävissä olleessa datajoukossa vaadittiin kuitenkin yllättävän usein digitaaliseen allekirjoitukseen perustuvia järjestelmäoikeuksia. Varmaa selitystä ilmiölle ei näyttäisi olevan. Eräs esimerkki tällaisesta oikeudesta on `BRICK`-oikeus. Tämän oikeuden tarkoitus on epäselvä: lähdekoodissa[7], jossa tämä oikeus on määritelty, mainitaan, että oikeutta voitaisiin käyttää laitteen käyttökelvottomaksi tekemiseen. Oikeuden tarkoituksesta on kuitenkin esitetty myös eriäviä ajatuksia.

Kuvassa 4.2 on kymmenen haitallisten sovellusten eniten vaatimaa allekirjoitussuojaustason järjestelmäoikeutta sekä hyvänlaatuisten sovellusten vastaavat määrät. Haitalliset sovellukset vaativat kahdeksassa kymmenestä tapauksesta enemmän kyseistä oikeutta kuin hyvänlaatuiset sovellukset.

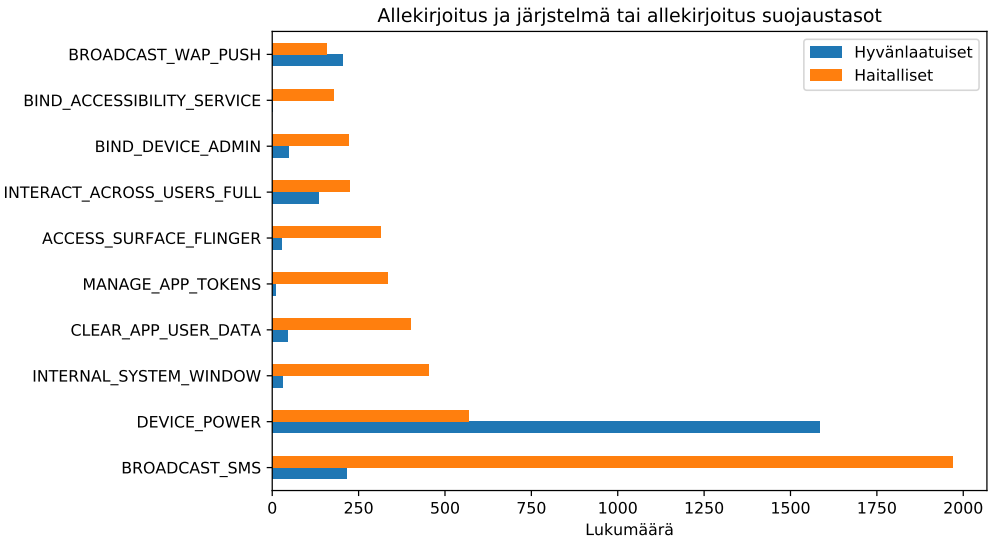
4.1 Muuttujavektorit ja niiden muodostus

Kaikista kerätyistä, sovellusten vaatimista järjestelmäoikeuksista muodostettiin binäärisiä vektoreita $v_i = (v_{i1}, v_{i2}, ..., v_{i276})$, jossa jokainen $v_{ik} = 1$, jos tapaus i

[7] https://git.io/vHHUx

Järjestelmäoikeus	Haitalliset	Hyvänlaatuiset
INTERNET	58 861	53 234
READ_PHONE_STATE	57 553	36 987
WRITE_EXTERNAL_STORAGE	56 206	42 172
ACCESS_NETWORK_STATE	55 037	47 729
ACCESS_WIFI_STATE	40 052	30 752
GET_TASKS	36 982	10 219
RECEIVE_BOOT_COMPLETED	33 660	19 110
INSTALL_SHORTCUT	32 949	10 319
WAKE_LOCK	32 608	37 421
SEND_SMS	30 793	10 376

Taulukko 2: Haitallisten sovellusten 10 eniten vaatimaa oikeutta ja hyvänlaatuisten sovellusten vastaavat määrät.



Kuva 4.2: Kymmenen haitallisten sovellusten eniten vaatimaa allekirjoitussuojaustason järjestelmäoikeutta sekä hyvänlaatuisten sovellusten vastaavat arvot.

vaatii oikeutta k ja muutoin $v_{ik} = 0$.

Vektorin v_i loppuun lisättiin myös luokkaleima sekä tapauksen yksilöivä tunniste. Tällä tavoin yhden tapauksen kuvaavaksi vektoriksi v_i saatiin 278-paikkainen binäärinen vektori, $v_i = (v_{i1}, v_{i2}, ..., v_{i276}, v_{ic}, v_{iuid})$, jossa v_{ic} on luokkaleima ja v_{iuid} on tapauksen yksilöllinen tunniste. Tunniste on hyödyllinen, mikäli on tarvetta tarkastella esimerkiksi syytä, miksi jokin sovellus tunnistettiin virheellisesti haitalliseksi. Tunnisteen avulla voidaan siis löytää kyseinen tapaus alkuperäisestä datajoukosta ja tarkemmin tarkastella sen muita ominaisuuksia.

4.2 Liittyvä tutkimus

Muussa aiheeseen liittyvässä tutkimuksessa on käytetty muuttujina vaadittujen ja käytettyjen oikeuksien (Moonsamy ym., 2014) lisäksi myös muun muassa matalan tason järjestelmäkutsuja (Burguera ym., 2011) ja modaliteeteiksi kutsuttuja sensitiivisten funktiokutsujen sekvenssejä (Yang ym., 2014).

Vaaditut oikeudet ovat kuitenkin hyvä lähtökohta käytännön tutkimukseen, sillä ne ovat suhteellisen helposti saatavilla eikä niitä voida salata (Moonsamy ym., 2014).

5 Menetelmät

Tässä luvussa tarkastellaan käytännön tutkimustyön työkaluja, menetelmiä ja vaiheita. Tutkimustyö koostui pääpiirteissään seuraavista vaiheista:

- muuttujien valitseminen
- muuttujien arvojen kerääminen
- muuttujavektoreiden muodostaminen
- luokittelu eri menetelmin
- tulosten tarkastelu.

Tutkielmassa käytettyinä muuttujina toimivat sovellusten vaatimat oikeudet, kuten luvussa 4 todettiin. Muuttujien arvot kerättiin datajoukosta tätä tarkoitusta varten toteutetulla `PHP`-ohjelmalla ja tallennettiin `JSON`-muodossa. Tämän jälkeen data muunnettiin binäärivektoreiksi ja samalla kerätyistä arvoista suodatettiin pois sellaiset oikeudet, jotka eivät kuulu Androidin määrittelemiin järjestelmäoikeuksiin. Lopuksi binäärivektorit tallennettiin tekstitiedostoiksi `CSV`-muodossa.

Luokittelussa käytettiin hyväksi Python-ohjelmointikieltä sekä sille toteutettua Scikit-Learn[8]-koneoppimiskirjastoa. Jatkossa viittaukset luokkien nimiin, metodien argumentteihin ja funktiokutsuihin viittaavat tämän koneoppimiskirjaston vastaaviin, ellei muuta ole asiayhteydessä erikseen mainittu. Myös argumenttien merkitykset löytyvät kyseisen koneoppimiskirjaston dokumentaatiosta.

Lisäksi muun muassa työn eri vaiheiden toistettavuuden helpottamiseksi käytettiin Jupyter Notebook -muistikirjasovellusta. Notebook on selaimessa toimiva sovellus, jossa voidaan yhdistellä Python-ohjelmakoodia, tekstiä ja grafiikkaa yhtenäiseksi muistikirjaksi. Muistikirjassa olevan koodin voi suorittaa, jolloin suoritetun koodin tulokset, kuten taulukot ja kuvaajat jäävät muistikirjaan näkyviin. Lisäksi lopullisen muistikirjan voi muuntaa muun muassa `PDF`-tiedostoksi. (Kluyver ym., 2016) Taulukosta 3 nähdään tutkielmassa käytetyt Scikit-Learn-ohjelmistokirjaston luokat selitteineen.

Varsinaiset koneoppimisalgoritmit toteuttavien luokkien lisäksi tutkimuksessa käytettiin myös `GridSearchCV`[9]-luokkaa, jonka avulla voidaan etsiä parasta hyperparametrien yhdistelmää. Kyseisen luokan rakentimen argumenteiksi annetaan jonkin algoritmin toteuttavan luokan instanssin lisäksi parametriavaruus, jonka kaikki eri kombinaatiot käydään lävitse sekä mallin hyvyyden arviointikriteeri.

Listauksessa 2 on esimerkki `GridSearchCV`-luokan käytöstä. Esimerkissä etsitään parasta k:n arvon ja etäisyysmitan yhdistelmää k:n lähimmän naapurin menetelmälle. Mahdolliset k:n arvot annetaan listana, joka luodaan Numpy-kirjaston

[8] http://scikit-learn.org

[9] http://scikit-learn.org/stable/modules/grid_search.html

Luokka	Selite
`KNeighborsClassifier`	K:n lähimmän naapurin luokittelija
`DecisionTreeClassifier`	Yksittäinen päätöspuu-luokittelija
`RandomForestClassifier`	Metsä päätöspuu-luokittelijoita
`GridSearchCV`	Hyperparametriyhdistelmän etsintä

Taulukko 3: Scikit-Learn-ohjelmistokirjastosta käytetyt luokat. Kolme ensimmäistä toteuttavat luokittelualgoritmit ja viimeinen on tarkoitettu parhaan hyperparametriyhdistelmän etsintään.

`arange`[10] -funktiolla. Funktion ensimmäinen argumentti kertoo arvoalueen alarajan, toinen ylärajan ja kolmas askelluksen. Esimerkin tapauksessa `arange`-funktiokutsu tuottaa taulukon $[3, 5, 7, 9, 11)$. Lisäksi etäisyysmitaksi annetaan kaksi eri vaihtoehtoa: Jaccard ja Manhattan.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

parameters = {
    'n_neighbors': np.arange(3, 11, 2),
    'metric': ['manhattan', 'jaccard']
}

estimator = KNeighborsClassifier()
grid = GridSearchCV(estimator, parameters, scoring='roc_auc')
grid.fit(X, y)
```

Listaus 2: Parhaan parametriyhdistelmän etsintä *k*:n lähimmän naapurin luokittelijalle. Mallin hyvyyttä arvioidaan `ROC-AUC`-arvolla.

Tutkielman yhteydessä testiajoja tehtiin myös muilla koneoppimismenetelmillä, kuten tukivektorikoneella (Support Vector Machine) sekä naiivilla Bayes-luokittelijalla (Naive Bayes Classifier). Näitä ei kuitenkaan eri syistä valittu tutkielmaan.

Tutkielmassa raportoidut testiajot toteutettiin henkilökohtaisella tietokoneella, jossa on 4 gigatavua `RAM`-muistia sekä 64-bittinen 2,8 gigahertsin kellotaajuu-

[10] https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html

della toimiva kaksiytiminen AMD:n prosessori. Tietokoneen käyttöjärjestelmänä oli niinikään 64-bittinen Windows 8.1 Pro.

Seuraavaksi kuvataan käytetyt menetelmät pääpiirteissään, niiden soveltaminen käytäntöön sekä saadut tulokset. Lisäksi verrataan saatuja tuloksia muuhun alan tutkimukseen.

5.1 *K*:n lähimmän naapurin luokittelumenetelmä

K:n lähimmän naapurin luokittelumenetelmä kuuluu niin sanottuihin prototyyppipohjaisiin menetelmiin. Menetelmässä ei luoda mallia opetusjoukon tapauksista, vaan uudelle testitapaukselle etsitään k lähintä tapausta eli naapuria opetusjoukosta. Luokiteltavan tapauksen luokkaleima määräytyy löydettyjen k:n naapurin luokkaleimojen enemmistön perusteella. (Louppe, 2014, s. 24)

Kuvassa 5.1 esitetään esimerkkitilanne, jossa on seitsemän opetusjoukon datapistettä ja yksi luokiteltava testitapaus (merkitty neliöllä) sekä valittu k:n arvoksi 3. Testitapauksen ympärille on piirretty ympyrä, jonka sisälle jää kolme lähintä tapausta. Kaksi kolmesta lähimmästä tapauksesta kuuluu tässä negatiiviseen luokkaan (merkitty ympyrällä) ja yksi kolmesta positiiviseen luokkaan (merkitty kolmiolla). Tällöin enemmistö kolmesta lähimmästä naapurista on negatiivisen luokan edustajia ja siten testitapaus luokitellaan negatiiviseen luokkaan. Tämän esimerkkitilanteen datapisteitä hyödynnetään myöhemmin listauksen 3 esimerkissä.

Kaksiluokkaisessa tapauksessa k:n arvoksi on hyvä valita jokin pariton kokonaisluku. Tällöin vältetään tasapeli-tilanne, jossa kummankin luokan edustajia on yhtä paljon. Tasapeli-tilanteessa luokkaleima jouduttaisiin ratkaisemaan esimerkiksi valitsemalla luokkaleima satunnaisesti. Mikäli luokkia on useampi kuin kaksi, tasapeli-tilannetta ei voida välttää valitsemalla k:n arvoksi pariton kokonaisluku. (Elkan, 2011, ss. 1-2; Hastie ym., 2009, s. 465)

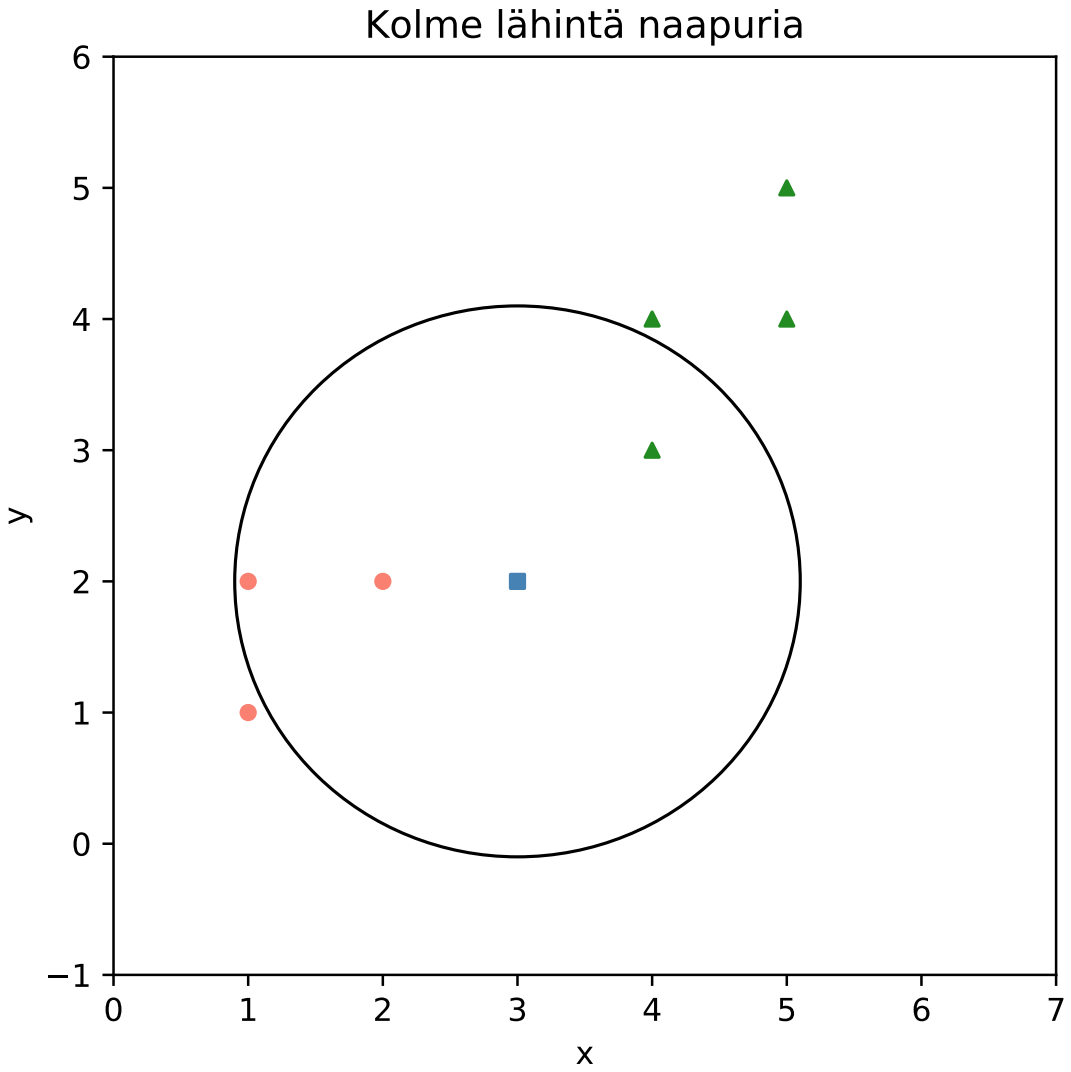
Tapausten läheisyyttä mitataan niin sanotulla euklidisella etäisyydellä. Euklidinen etäisyys kahden tapauksen välillä määritellään kaavalla

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}, \tag{1}$$

jossa X_1 ja X_2 ovat siis tapauksia, joiden välistä etäisyyttä mitataan. (Han ym., 2006, ss. 348-349)

Euklidista etäisyyttä käytettäessä on tärkeää huomioida muuttujien määrittelyväli. Mikäli jokin muuttuja x_i on määritelty välillä $[0, 1]$ ja vastaavasti toinen muuttuja x_j välillä $[0, 10^6]$, niin muuttuja x_j dominoi etäisyyksien laskennassa. Tällöin muuttuja x_i jää huomioimatta. Tästä syystä kaikki muuttujat skaalataan samalle arvovälille, esimerkiksi $[0, 1]$. Skaalaamiseen voidaan käyttää esimerkiksi min-max normalisointia tai standardointia. (Han ym., 2006, ss. 348-349; James ym., 2013, s. 165)

Vaikka k:n lähimmän naapurin menetelmä on yksinkertainen, sen on todettu toimivan hyvin varsinkin sellaisissa tilanteissa, joissa luokkarajat ovat monimutkaisia. (Louppe, 2014, s. 24) Menetelmän ongelmana on kuitenkin sen laskennallinen vaativuus, nimittäin $\mathcal{O}(N)$ yhdelle luokiteltavalle tapaukselle (Han



Kuva 5.1: Esimerkkitilanne, jossa k:n arvoksi on valittu 3. Ympyrällä merkityt (punaiset) tapaukset kuuluvat negatiiviseen luokkaan ja kolmiolla merkityt (vihreät) positiiviseen luokkaan. Neliöllä merkitty (sininen) on uusi testitapaus.

ym., 2006, s. 349). Muuttujien määrä voi olla huomattava (Breiman, 2001, s. 6) ja siksi myös se on hyvä huomioida. Tällöin vaativuudeksi saadaan $\mathcal{O}(dN)$, jossa d on muuttujien määrä. Tietyissä tilanteissa lähimmän naapurin etsintää voidaan nopeuttaa käyttämällä raa'an voiman haun sijaan puurakenteita. (Elkan, 2011, s. 2)

5.1.1 Käytännön sovellutus ja tulokset

Koska k:n lähimmän naapurin menetelmä on laskennallisesti raskas, ei testiajoissa voitu hyödyntää koko käytettävissä olevaa datajoukkoa. Opetusjoukoksi valittiin aluksi satunnaisesti 20% tapauksista eli noin 12 000 tapausta molemmista luokista, yhteensä siis noin 24 000 tapausta. Valinnassa säilytettiin luokkajakau-

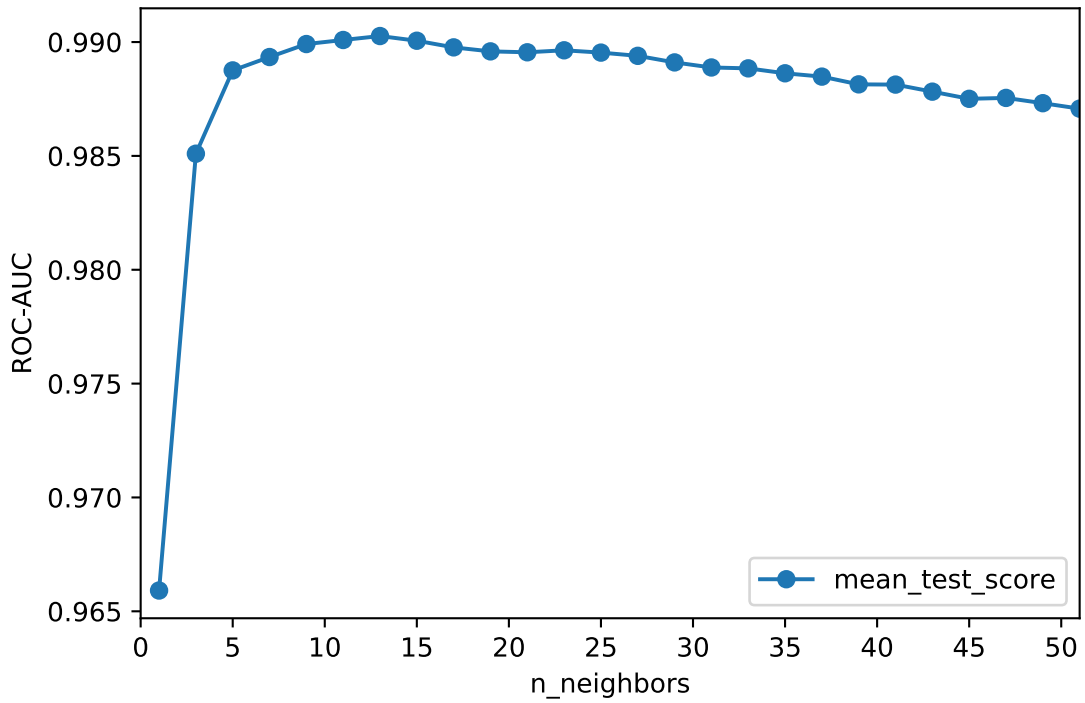
ma eli osajoukko sisälsi suhteellisesti saman osuuden kumpaakin luokkaa kuin koko datajoukko.

Parhaan *k*:n arvon etsintä

Testiajossa etsittiin ensin `GridSearchCV`-luokkaa hyödyntäen parasta *k*:n arvoa. Parittomat *k*:n arvot valittiin väliltä $[1, 52]$, jolloin mahdollisia arvoja on yhteensä 26 kappaletta. `GridSearchCV`-luokka käyttää oletusarvoisesti kolminkertaista ristiinvalidointia[11], joten erilaisia testiajoja kertyi tällä tavoin yhteensä

$$3 \times 26 = 78.$$

Kuvassa 5.2 on esitetty `ROC-AUC`-arvo *k*:n arvon funktiona. Kuten kuvasta voidaan nähdä, *k*:n arvolla 1 `ROC-AUC`-arvo on huomattavasti alhaisempi kuin muilla *k*:n arvoilla. Kun *k*:n arvoa kasvatetaan, `ROC-AUC`-arvo kasvaa voimakkaasti tiettyyn pisteeseen saakka, jonka jälkeen se lähtee laskemaan lievästi.



Kuva 5.2: `ROC-AUC`-arvo *k*:n arvon funktiona.

Kuvan perusteella voidaan arvioida, että `ROC-AUC`-arvo saavuttaa maksimiarvonsa *k*:n arvon ollessa 13. Tämän voi vahvistaa myös `GridSearchCV`-luokan palauttamista tilastoista. Taulukossa 4 on kolme parasta *k*:n arvoa ja niitä vastaavat `ROC-AUC`-arvot, jotka ovat hyvin lähellä toisiaan.

Kuten taulukosta 4 nähdään, seuraavaksi parhaat *k*:n arvot ovat 11 ja 15. Näin kolme parasta *k*:n arvoa ovat 13, 11 ja 15. Samanlaisen tuloksen antoi myös toinen

[11] http://scikit-learn.org/stable/modules/grid_search.html

ROC-AUC	k
0.9903	13
0.9901	11
0.9901	15

Taulukko 4: Kolme parasta k:n arvoa sekä niitä vastaavat ROC-AUC-arvot.

testiajo, joskin sama kolmen kärki oli hieman eri järjestyksessä.

Kaikkien k:n naapurin on oltava saman luokan edustajia

Testiajoissa kokeiltiin myös ajatusta siitä, että uuden tapauksen luokittelemiseen haitalliseksi vaadittaisiin kaikkien k:n lähimmän naapurin haitallisuus[12]. Tutkielmassa käytetyssä ohjelmistokirjastossa tämä voidaan toteuttaa `predict_proba`[13]-metodin avulla. Metodi ennustaa testitapauksen todennäköisyydet kuulua annettuihin luokkiin listauksen 3 osoittamalla tavalla. Listauksen esimerkissä testitapaus[14] kuuluu noin 67% todennäköisyydellä negatiiviseen luokkaan ja vastaavasti noin 33% todennäköisyydellä positiiviseen luokkaan. Tällöin testitapaus luokiteltaisiin negatiiviseen luokkaan, koska sen kaikki naapurit eivät kuulu samaan positiiviseen luokkaan.

Kun halutaan kaikkien k:n lähimmän naapurin olevan samaa luokkaa, on kyseisen luokan todennäköisyyden oltava 100%. Tällöin kaikki ne tapaukset, jotka tavallisesti luokiteltaisiin positiiviseen luokkaan todennäköisyydellä

$$0.5 < P(y = 1) < 1.0, \tag{2}$$

luokitellaankin nyt negatiiviseen luokkaan. Positiivisen luokkaleiman saavat silloin ainoastaan ne tapaukset, jotka ovat positiivisia todennäköisyydellä

$$P(y = 1) = 1.0. \tag{3}$$

Listauksessa 4 esitetään edellä kuvatun menetelmän toteutuksen pääkohdat. Nyt `predict_proba`-metodi palauttaa $n \times 2$ -kokoisen taulukon, jossa ensimmäinen sarake sisältää tapauksen todennäköisyyden kuulua negatiiviseen luokkaan ja toinen sarake todennäköisyyden kuulua positiiviseen luokkaan. Tällöin voidaan toisessa sarakkeessa merkitä nolliksi ne elementit, joiden arvo on pienempi kuin yksi ja käyttää näin menettelemällä saadun sarakkeen arvoja ennustettuina luokkaleimoina.

[12] F-Securen henkilöstö esitti ajatuksen kyseisestä menetelmästä kirjoittajan keskustellessa heidän kanssaan tutkielman teon aikana (F-Secure, ei julkaisupäivää[b]).

[13] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[14] Datapisteitä ja esimerkin tilannetta on havainnollistettu kuvassa 5.1.

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

X = pd.DataFrame([
    [1, 1], [1, 2], [2, 2],
    [4, 3], [4, 4], [5, 4],
    [5, 5]
])

y = pd.Series([0, 0, 0, 1, 1, 1, 1])

test = np.array([3, 2]).reshape(1, -1)

classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X, y)

y_proba = classifier.predict_proba(test)
print(y_proba) # [[ 0.66666667  0.33333333]]
```

Listaus 3: Esimerkki `predict_proba`-metodin käytöstä. Testitapaus kuuluu noin 67% todennäköisyydellä negatiiviseen luokkaan eli luokkaan 0.

Testiajo suoritettiin viisinkertaisella ristiinvalidoinnilla. Opetusjoukoksi valittiin satunnaisesti 20% koko opetusjoukosta, jonka lisäksi testijoukoksi valittiin satunnaisesti jäljelle jääneestä joukosta 20%. Molemmat joukot olivat lähes yhtä suuria.

Vaatimus, että kaikkien k:n lähimmän naapurin on oltava haitallisia tapauksen luokittelemiseksi haitalliseksi näyttäisi toimivan sekä euklidisella että Manhattan- etäisyydellä. Muilla käytetyillä etäisyysmitoilla, kuten esimerkiksi Jaccard-etäisyysmitalla tulokset olivat huonompia. Taulukossa 5 on eräs edellä kuvatun testiajon tuottama sekaannusmatriisi. Sekaannusmatriisista saadaan hyvin alhainen väärien positiivisten osuus, noin 0.0014 eli likimain 0.15% ja luokittelijan tarkkuudeksi noin 0.93 eli likimain 93%.

Menetelmän laskennallinen vaativuus.
Menetelmän laskennallinen vaativuus tuli käytännössä esille testiajojen aikana. Esimerkiksi edellä kuvattu testiajoryhmä, jossa etsittiin parasta k:n arvoa, kesti

```
from sklearn.neighbors import KNeighborsClassifier

estimator = KNeighborsClassifier()
estimator.fit(X_train, y_train)

# ...

probabilities = estimator.predict_proba(X_test)
probabilities[probabilities[:, 1] < 1.0, 1] = 0
print(confusion_matrix(y_test, probabilities[:, 1]))
```

Listaus 4: Nyt k:n lähimmän naapurin on oltava haitallisia, jotta tapaus luokitellaan haitalliseksi eli positiiviseksi. Viimeinen rivi tulostaa sekaannusmatriisin.

		Ennuste	
		Negatiivinen	Positiivinen
Todellinen	Negatiivinen	9784	14
	Positiivinen	1367	8470

Taulukko 5: Sekaannusmatriisi k:n lähimmän naapurin luokittelusta, jossa kaikkien k:n lähimmän naapurin on oltava haitallisia, jotta uusi tapaus luokitellaan haitalliseksi.

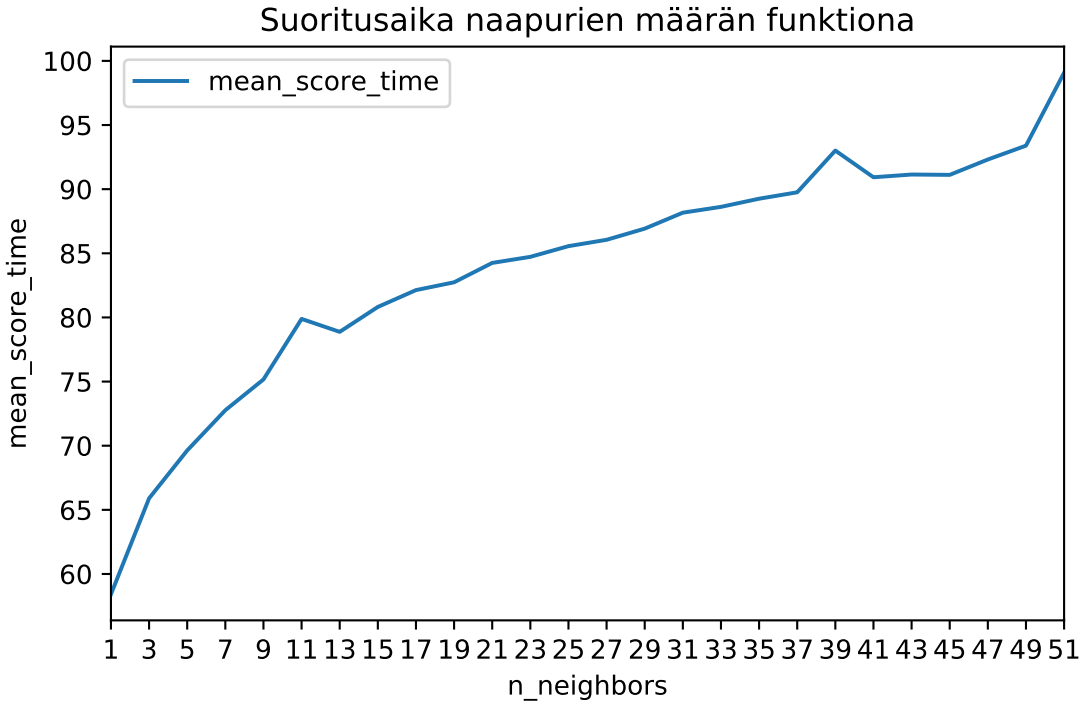
lähes kolme tuntia[15]. Keskimääräinen ajoaika yhdelle testiajolle vaikutti nousevan k:n arvon kasvattamisen myötä. Kuvassa 5.3 on kuvattu keskimääräiset ajoajat k:n funktiona. Kuvasta voidaan havaita, että k:n arvon ollessa 1, ajoaika on keskimäärin hieman alle minuutin, mutta esimerkiksi parhaalla k:n arvolla ajoaika on noussut jo lähes puoleentoista minuuttiin.

5.1.2 Liittyvä tutkimus

Sanz ym. (Sanz ym., 2013) käyttivät tutkimuksessaan k:n lähimmän naapurin luokittelijaa yhtenä koneoppimismenetelmistä. Lisäksi he käyttivät `ROC-AUC`-arvoa luokittelijan hyvyyden arviointiin. Testiajoissa heidän toisena muuttujajoukkona olivat sovellusten vaatimat oikeudet.

Sanzin ym. testiajojen asettelu ja parametrit olivat samankaltaiset kuin tässä tutkielmassa, joten tuloksia on tässä mielessä mahdollista verrata keskenään. Par-

[15]Käytetyn laitteiston tiedot on lueteltu edellä, luvun 5 lopussa.



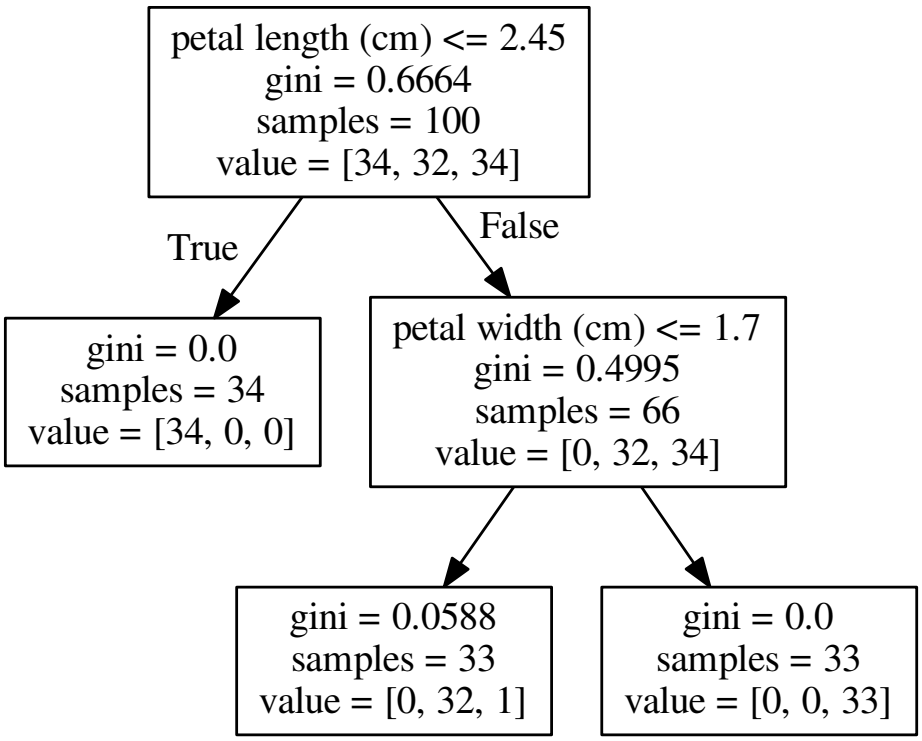
Kuva 5.3: Keskimääräinen ajoaika *k*:n funktiona.

haimmat tulokset he saivat *k*:n arvoilla 1 ja 3. Sanzin ym. datajoukko oli kuitenkin erittäin pieni: noin 330 tapausta sekä haitallisia että hyvänlaatuisia, yhteensä vain alle 700 tapausta. Määrä oli siis huomattavasti vähemmän kuin tässä tutkielmassa.

5.2 Päätöspuu

Päätöspuut ovat joukko puurakennetta hyödyntäviä koneoppimismenetelmiä. Päätöspuiden idea esiteltiin jo 1960-luvun taitteessa muun muassa Morganin ja Sonquistin toimesta, mutta pääasiallisen kehitystyön tekivät myöhemmin Breiman ja Quinlan. (Louppe, 2014, s. 25) Geurts toteaa myös, että vaikka puupohjaiset menetelmät olivat jo aiemmin tunnettuja, vasta Breimanin ym. kirja "Classification And Regression Trees" teki puumenetelmistä suositun (Geurts, 2002, s. 87).

Kuvassa 5.4 on esimerkki eräästä päätöspuusta Iris-datalle. Kuvan esimerkissä päätöspuun maksimisyvyys oli rajoitettu kahteen, mutta muiden argumenttien arvoina olivat niiden oletusarvot. Oletusarvoja on listattu edempänä taulukossa 6. Esimerkin päätöspuun tarkkuus oli erillisellä testijoukolla 90%.



Kuva 5.4: Eräs päätöspuu Iris-datalle. Päätöspuun maksimisyvyys on tässä tapauksessa 2.

Päätöspuiden hyviin ominaisuuksiin kuuluvat muun muassa niiden nopeus (Geurts, 2002, katso kappale 1.1.4.), sisäänrakennettu muuttujien valinta, sekä robustius kohinaisille muuttujille ja poikkeaville tapauksille (Louppe, 2014, s. 26). Lisäksi esimerkiksi Geurts (Geurts, 2002, s. 4) ja Louppe (Louppe, 2014, s. 26)

ovat esittäneet päätöspuun tulkittavuutta tai luettavuutta puoltavia näkemyksiä. Puurakenne näyttääkin helposti tulkittavalta, mutta menetelmän epävakauden vuoksi tulkintojen tekemisen suhteen tulee olla varovainen (Mohri ym., 2014, s. 198).

Päätöspuun ongelmina ovat muun muassa *ylioppiminen* (overfitting) sekä *epävakaus* (instability) (Scikit Learn, 2016). Ylioppimisella tarkoitetaan sitä, että malli oppii opetusjoukon hyvin, mutta sillä ei ole kykyä yleistää. Rokachin ym. mukaan päätöspuiden yhteydessä ylioppimiseen voidaan vaikuttaa kahdella eri tavalla: jättämällä hyödyttömät jaot tekemättä sekä jälkikarsinnalla. Hyödyttömästä jaosta Rokach mainitsee esimerkkinä vain tilastollisesti merkittävien jakojen tekemisen. (Rokach ja Maimon, 2007, s. 49-50) Ylioppimiseen voidaan vaikuttaa myös muodostamalla päätöspuista satunnaismetsä (Breiman, 2001, ss. 6, 29), joita tarkastellaankin luvussa 5.3.

Puu on myös graafin erikoistapaus, jossa kaarien lukumäärä $m = n - 1$, jossa n tarkoittaa graafin solmujen lukumäärää (Koivisto ja Niemistö, 2001, ss. 60-61). Tämän vuoksi päätöspuut voidaan lukea laajempaan induktiograafien joukkoon. (Louppe, 2014, s. 29) Induktiograafeja ei tässä tutkielmassa käsitellä, mutta lukija voi niin halutessaan tutustua esimerkiksi alaviitteessä[16] mainittuihin lähteisiin.

5.2.1 Päätöspuun kasvattaminen

Päätöspuun kasvattamiseen tarkoitettuja algoritmeja on useampia erilaisia (Geurts, 2002, s. 76). Esimerkkeinä näistä algoritmeista mainittakoon Breimanin ym. `CART` (Breiman ym., 1984) sekä Quinlanin `ID3` (Quinlan, 1986) ja `C4.5` (Quinlan, 1993). Tässä tutkielmassa käytetyssä Scikit-Learn-ohjelmistokirjastossa on toteutettu optimoitu `CART`-algoritmi (Scikit Learn, 2016).

Päätöspuu sisältää erilaisia solmuja: *juurisolmun*, *sisäsolmuja* sekä *lehtisolmuja*. Päätöspuun lehtisolmua sanotaan puhtaaksi, jos se sisältää vain yhden luokan tapauksia ja vastaavasti epäpuhtaaksi muussa tapauksessa. (Raileanu ja Stoffel, 2004, s. 79) Ideaalinen tilanne olisi se, että kaikki samaan solmuun päätyvät tapaukset olisivat samaan luokkaan kuuluvia eli solmu olisi puhdas, sillä puhtaampi solmu tuottaa paremman ennusteen tapaukselle (Duda ym., 2001, s. 396) (Louppe, 2014, s. 30). Näin ei kuitenkaan yleensä ole, vaan solmussa on useampaa eri luokkaa edustavia tapauksia. Tällöinkin pyritään siihen, että solmut olisivat mahdollisimman puhtaita (Duda ym., 2001, s. 396, 398).

Puun jokainen solmu edustaa opetusjoukon osajoukkoa (Breiman ym., 1984, s. 22). Päätöspuu kasvatetaan jakamalla solmu t kahteen (tai useampaan) lapsisol-

[16] Induktiograafeista kirjoittavat muun muassa Zighed ym. (Zighed ja Rakotomalala, 2000) sekä Kohavi ym. (Kohavi ja Li, 1995). Ensimmäinen näistä kahdesta on kirjoitettu ranskaksi ja jälkimmäinen englanniksi.

muun t_v ja t_o siten, että syntyvien lapsisolmujen epäpuhtaus on mahdollisimman pieni verrattuna solmun t epäpuhtauteen. (Louppe, 2014, s. 30)

Epäpuhtauden muutos lapsisolmujen ja vanhemman solmun välillä määritellään yleisesti binäärisen puun tapauksessa seuraavasti[17]:

$$\Delta i(s,t) = i(t) - p_v i(t_v) - p_o i(t_o) \tag{4}$$

jossa $i(t)$ on jokin funktio epäpuhtauden arviointiin, p_v ja p_o ovat vasempaan ja oikeaan lapsisolmuun kuuluvien tapausten suhteelliset osuudet. (Louppe, 2014, ss. 30-31)

Breimanin ym. `CART`-algoritmissa valittiin funktioksi $i(t)$ Gini-indeksi (Breiman ym., 1984, s. 103). Tutkielmassa käytetyssä päätöspuun toteutuksessa on kuitenkin mahdollista käyttää kumpaakin, sekä Gini-indeksiä että Shannonin entropiaa[18]. Tarkastellaan seuraavaksi lyhyesti edellä mainittuja kriteereitä.

Gini-indeksi (*Gini index*) on alun perin italialaisen tilastotieteilijä Corrado Ginin esittelemä indeksi. Tätä ei tule sekoittaa Gini-kertoimeen (*Gini coefficient*), joka on myös saman tilastotieteilijän esittelemä kerroin. (Flach, 2012, s. 134, katso lähteen alaviite) Gini-indeksi määritellään solmussa t kaavalla[19]

$$i_G(t) = \sum_{k=1}^{J} p(c_k|t)(1 - p(c_k|t)). \tag{5}$$

Entropia puolestaan on Claude Shannonin (Shannon, 1948) vuonna 1948 esittelemä informaatioteoreettinen käsite. Entropia määritellään solmussa t kaavalla[20]

$$i_H(t) = -\sum_{k=1}^{J} p(c_k|t) \log_2 p(c_k|t). \tag{6}$$

Epäpuhtauden muutos lasketaan solmulle t kaavalla[21]

$$\Delta i(s,t) = i(t) - p_v i(t_v) - p_o i(t_o). \tag{7}$$

Kun funktiona $i(t)$ käytetään Shannonin entropiaa, arvoa $\Delta i_H(s,t)$ nimitetään *informaatiohyödyksi* (information gain) (Louppe, 2014, s. 45).

Tutkielmassa käytetty päätöspuun toteutus sisältää myös mahdollisuuden arvioida muuttujien tärkeyttä. Valmiiksi kasvatetun puun `feature_importances_`-attribuutti on taulukko, joka sisältää muuttujien tärkeysarvot.[22] Muuttujien tärkeysarvoihin palataan vielä satunnaismetsien yhteydessä luvussa 5.3.

[17] Epäpuhtauden muutoksen kaava on (lähes) sama kuin Louppen (Louppe, 2014, s. 30) väitöskirjassa.

[18] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[19] Gini-indeksin kaava on sama kuin Louppen (Louppe, 2014, s. 45) väitöskirjassa.

[20] Shannonin entropian kaava on sama kuin Louppen (Louppe, 2014, s. 45) väitöskirjassa.

[21] Epäpuhtauden muutoksen kaava on sama kuin Louppen (Louppe, 2014, s. 30) väitöskirjassa.

[22] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

5.2.2 Käytännön sovellutus ja tulokset

Päätöspuilla suoritettiin useita erilaisia testiajoja. Seuraavaksi tarkasteltavat testiajot jakautuvat kahteen eri testiajoryhmään. Ensimmäisessä ryhmässä käytettiin 10-kertaista ristiinvalidointia sekä koko käytettävissä olevaa datajoukkoa. Hyperparametrien arvoina olivat tutkielmassa käytetyn ohjelmistokirjaston päätöspuualgoritmin toteuttavan luokan argumenttien oletusarvot. Osa näistä argumenteista on listattu taulukossa 6.

Argumentti	Oletusarvo
`criterion`	gini
`max_depth`	`None`
`min_samples_split`	2
`min_samples_leaf`	1
`max_leaf_nodes`	`None`
`max_features`	`None`

Taulukko 6: Päätöspuun toteuttavan luokan argumentteja oletusarvoineen.

Argumenteista esimerkiksi `min_samples_leaf`[23] määrittelee, kuinka monta tapausta solmussa tulee olla, että se voi olla lehtisolmu. Lehtisolmun tapausten määrää voidaan kontrolloida myös `min_samples_split`[24]-argumentin avulla, joka määrittelee, kuinka monta tapausta solmussa tulee olla, jotta solmu voidaan jakaa edelleen vasempaan ja oikeaan lapsisolmuun. Vaatimalla, että lehtisolmussa tulee olla enemmän kuin yksi tapaus, vähennetään ylioppimisen riskiä (Scikit Learn, 2016).

Taulukon 6 viimeisellä rivillä on esitelty `max_features`[25]-argumentti, joka määrittelee, kuinka monen eri muuttujan joukosta parasta jakomuuttujaa etsitään. Päätöspuun kohdalla oletusarvo on `None`, mikä tarkoittaa käytännössä sitä, että arvoa ei ole annettu ja sen vuoksi käytetään oletusarvoisesti kaikkia muuttujia. Tämä on tärkeä huomata, sillä sama argumentti esiintyy myös satunnaismetsän yhteydessä, jossa sen oletusarvona on `auto`. Tällöin käytettävä muuttujien määrä on oletusarvoisesti kaikkien muuttujien lukumäärän neliöjuuri[26].

Kymmenkertainen ristiinvalidointi toteutettiin ohjelmistokirjaston tarjoamaa `cross_val_predict`[27]-funktiota käyttäen. Funktiolle annetaan argumentteina luokittelijan eli tässä tapauksessa päätöspuun instanssi, opetusjoukko sekä luokkaleimat ja ristiinvalidoinnin määrittelevä argumentti.

[23] Katso alaviite 22

[24] Katso alaviite 22.

[25] Katso alaviite 22.

[26] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[27] http://scikit-learn.org/stable/modules/generated/sklearn.model$_s$*election.cross*$_v$*al*$_p$*redict.html*

Kun `cross_val_predict`-funktiota kutsutaan, se suorittaa ristiinvalidoinnin ja palauttaa jokaiselle tapaukselle luokituksen. Luokituksista ja todellisista luokkaleimoista voidaan muodostaa sekaannusmatriisi, josta nähdään luokittelijan tehokkuus eli miten hyvin tai huonosti luokittelija toimii.

Sekaannusmatriisin arvoista voidaan laskea erilaisia luokittelijan – tässä tapauksessa päätöspuun – hyvyyden arviointiin käytettäviä tunnuslukuja, kuten *väärien positiivisten suhteellinen määrä*[28] (False Positive Rate) ja *väärien negatiivisten suhteellinen määrä*[29] (False Negative Rate) sekä *tarkkuus* (accuracy)[30].

Taulukossa 7 on erään edellä kuvatun testiajon tuottama sekaannusmatriisi. Vääriä positiivisia oli tässä tapauksessa 828 kappaletta ja vääriä negatiivisia 1521 kappaletta.

		Ennuste	
		Negatiivinen	Positiivinen
Todellinen	Negatiivinen	60 414	828
	Positiivinen	1 521	59 960

Taulukko 7: Eräs päätöspuun oletusasetuksillaan tuottama sekaannusmatriisi.

Väärien positiivisten suhteellinen osuus voidaan laskea seuraavalla kaavalla:

$$FPR = \frac{FP}{FP + TN}, \tag{8}$$

jossa FP on väärien positiivisten määrä ja TN on todellisten negatiivisten määrä.

Sijoitettaessa kaavaan tarvittavat arvot sekaannusmatriisista (Taulukko 7), saadaan väärien positiivisten suhteelliseksi osuudeksi noin 0.0135 eli noin 1.4 prosenttiyksikköä.

Sekaannusmatriisista voidaan laskea myös luokittelijan tarkkuus. Tarkkuus lasketaan kaavalla

$$ACC = \frac{TP + TN}{N}, \tag{9}$$

jossa N on kaikkien tapausten määrä, TP on todellisten positiivisten määrä ja TN on todellisten negatiivisten määrä. Kun kaavaan sijoitetaan tarvittavat arvot sekaannusmatriisista (Taulukko 7), saadaan päätöspuun tarkkuudeksi likimain 0.9810 eli hieman yli 98 prosenttiyksikköä.

Toisessa testiajoryhmässä etsittiin ensin parasta arvoa puun maksimisyvyydelle `GridSearchCV`-luokan avulla. Tämän jälkeen kasvatettiin uusi päätöspuu

[28] https://en.wikipedia.org/wiki/Sensitivity_and_specificity

[29] Katso alaviite 28.

[30] Katso alaviite 28.

käyttäen hyväksi löydettyä parasta päätöspuun maksimisyvyyden arvoa. Lopuksi tehtiin vielä kymmenkertainen ristiinvalidointi tällä päätöspuulla.

Parasta arvoa päätöspuun maksimisyvyydelle etsittiin väliltä $5 \leq x < 45$, jossa x on siis maksimisyvyys. Kaikkia arvoja ei edellä määritellyltä väliltä käyty lävitse, vaan arvoja valittiin kahden välein eli ensimmäinen arvo oli 5, seuraava 7 ja niin edelleen.

Taulukossa 8 on eräs edellä kuvatun toisen testiajoryhmän päätöspuun tuottama sekaannusmatriisi. Kuten taulukosta voidaan havaita, tapauksista vain 10% jätettiin testiajoon, joka suoritettiin hyperparametrin etsinnän jälkeen.

		Ennuste	
		Negatiivinen	Positiivinen
Todellinen	Negatiivinen	6050	75
	Positiivinen	146	6002

Taulukko 8: Eräs päätöspuun tuottama sekaannusmatriisi toisesta testiajo-ryhmästä.

Vaikka päätöspuu antoi suhteellisen hyviä tuloksia, on syytä kuitenkin huomioida päätöspuiden epävakaus: pienetkin muutokset opetusjoukossa voivat saada aikaan aivan erilaisen päätöspuun (Rokach ja Maimon, 2007, s. 106).

5.2.3 Liittyvä tutkimus

Yerima ym. (Yerima ym., 2014) käyttivät tutkimuksessaan myös päätöspuuta yhtenä viidestä eri algoritmista. Heidän tutkimuksessaan algoritmeja testiajettiin ensin erikseen ja sitten myös näiden viiden algoritmin yhdistelmänä. Yksittäin ajettuna päätöspuu oli `ROC-AUC`-arvolla mitattuna keskimmäinen viidestä eri algoritmista.

Yeriman tulokset eivät kuitenkaan ole suoraan verrattavissa tämän tutkielman tuloksiin, sillä datajoukon koot ovat huomattavan erilaiset. Sen lisäksi Yerima ym. käyttivät myös muita ominaisuuksia vaadittujen oikeuksien lisäksi. Heidän valintansa käyttää päätöspuuta osoittautui kuitenkin hyväksi, sillä `ROC-AUC`-arvo oli hieman yli 0.96 jääden heidän parhaalle menetelmälleen noin 1,5%.

5.3 Satunnaismetsä

Yksi päätöspuiden iso ongelma on niiden epävakaus. Epävakaudesta kertoo herkkyys pienillekin muutoksille opetusjoukossa. Tällöin nämä pienet opetusjoukon muutokset saattavat muuttaa kasvatetun päätöspuun topologiaa eli puun rakennetta. Epävakaudesta johtuvia ongelmia on pyritty ratkaisemaan yhdistämällä useita toisistaan eroavia päätöspuita yhdeksi luokittelijoiden *yhdistelmäksi* (ensemble). (Rokach ja Maimon, 2007, ss. 47-48,106)

Esimerkiksi Breimanin kuvailema *bagging* (lyhennelmä sanoista bootstrap aggregation) on eräs yhdistelmämenetelmien ryhmä. Menetelmän ideana on yhdistää useita epävakaita luokittelijoita yhdeksi luokittelijaksi. Tällöin yhdistelmäluokittelijan tuloksen antavat kaikki yksittäiset luokittelijat yhdessä, esimerkiksi äänestämällä tai keskiarvoistamalla. (Breiman, 1996, ss. 123-124,135)

Bagging-menetelmän yksittäiset luokittelijat rakennetaan opetusjoukosta otetuista satunnaisotoksista palauttaen. Jokainen satunnaisotos on kooltaan yhtä suuri kuin koko opetusjoukko. Tällaista satunnaisotosta palauttaen kutsutaan termillä *bootstrap-otos* (bootstrap sample). (Breiman, 1996, ss. 123-124,136)

Bagging-menetelmässä uuden tapauksen luokkaleima määräytyy (enemmistö)äänestyksen perusteella (Breiman, 1996, s. 123). Uuden luokiteltavan tapauksen luokkaleimaksi asetetaan se, joka toistuu useimmin yksittäisten luokittelijoiden antamissa tuloksissa (James ym., 2013, s. 317). Luokkaleima voi määräytyä myös muulla tavalla. Esimerkiksi tutkielmassa käytetyssä ohjelmistokirjastossa lasketaan luokille todennäköisyyksien keskiarvot[31], joiden perusteella määräytyy luokiteltavan tapauksen luokkaleima.

Breimanin kehittämä satunnaismetsä perustuu edellä esitettyyn bagging-menetelmään. Satunnaismetsässä bagging-menetelmään on yhdistetty myös ominaisuuksien osajoukon satunnainen valinta. (Breiman, 2001, ss. 11, 27) Tällöin metsän jokainen päätöspuu kasvatetaan opetusjoukon suuruisesta bootstrap-otoksesta ja jokaisessa puun solmussa paras muuttuja valitaan muuttujien satunnaisesti valitusta osajoukosta. (Breiman, 1996, s. 123; Breiman, 2001, s. 11)

Satunnaismetsät ovat nopeita (Rokach ja Maimon, 2007, s. 109). Metsän yksittäisen puun kasvattaminen on nopeampaa kuin yksittäistä päätöspuualgoritmia käytettäessä, sillä metsän puiden kasvattamiseen käytetään vain muuttujien osajoukkoa (Breiman, 2001, s. 14). Jokaisessa solmussa tehdään määrätyn kokoinen satunnaisotos käytettävissä olevista muuttujista ja parasta jakomuuttujaa etsitään tästä satunnaisesti valitusta osajoukosta (Breiman, 2001, s. 11). Satunnaismetsän puiden kasvattaminen voidaan myös helposti rinnakkaistaa. Tällöin useita puita voidaan kasvattaa samanaikaisesti, mikä nopeuttaa metsän kasvattamista (Brei-

[31] http://scikit-learn.org/stable/modules/ensemble.html

man, 1996, s. 135).

Jaossa käytettävien muuttujien määrää kontrolloidaan `max_features`-argumentilla, jonka oletusarvo on kaikkien muuttujien lukumäärän neliöjuuri[32]. Koska neliöjuuri voi olla myös muu kuin kokonaisluku, tutkielmassa käytetyssä ohjelmistokirjastossa otetaan vain neliöjuuren arvosta kokonaislukuosa[33]. Tutkielman datajoukossa oli 276 muuttujaa, josta neliöjuuri on likimain 16.6, jolloin `max_features`-argumentin arvoksi tulee 16.

Opetusjoukossa voi myös olla useampia muuttujia, jotka ovat lähes yhtä hyviä kuin paras muuttuja. Tällöin yksittäistä päätöspuuta rakennettaessa paras muuttuja *peittää* (mask) muut hyvät muuttujat. Satunnaismetsässä, kun muuttujan valinta tehdään kaikkien muuttujien osajoukosta, myös nämä muut hyvät muuttujat saavat mahdollisuuden tulla valituiksi parhaan muuttujan sijaan ja tällöin peittymisilmiö vähenee. (Louppe, 2014, ss. 124-125)

5.3.1 Käytännön sovellutus ja tulokset

Satunnaismetsillä suoritettiin testiajoja laajasti, laajemmin kuin muilla tutkielmassa käytetyillä menetelmillä. Viimeisimmissä testiajoissa ajettiin testiajoa 361 erilaisella satunnaismetsällä, joista jokaiseen käytettiin 3-kertaista ristiinvalidointia. Näistä tulee siis yhteensä

$$3 \times 361 = 1083 \tag{10}$$

erilaista testiajoa. Tarkastellaan seuraavaksi saatuja tuloksia.

Listauksessa 5 on ote koodista, jota käytettiin testiajossa. Listauksessa etsitään satunnaismetsälle parasta puun maksimisyvyyden ja puiden lukumäärän yhdistelmää. Listauksessa nähtävä `arange`[34]-funktio luo taulukon, jossa on arvoja puoliavoimelta väliltä $[a, b)$ eli arvojoukko koostuu arvoista välillä

$$a \leq x < b. \tag{11}$$

Tälle funktiolle voidaan antaa myös kolmas argumentti, joka kertoo askelluksen.

Maksimisyvyyden tapauksessa askellus on kaksi, jolloin maksimisyvyyttä etsitään väliltä

$$2 \leq x < 40, \tag{12}$$

x:n ollessa aina parillinen. Vastaavasti metsän puiden lukumäärää etsitään väliltä

[32] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[33] https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/tree/tree.py

[34] https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# ...

parameters = {
    'max_depth': np.arange(2, 40, 2),
    'n_estimators': np.arange(10, 200, 10),
}

estimator = RandomForestClassifier()
grid = GridSearchCV(estimator, parameters, scoring='roc_auc')
grid.fit(X_train, y_train)
```

Listaus 5: Parhaan hyperparametri-yhdistelmän etsintää satunnaismetsälle. Etsittävinä parametreina puun syvyys ja puiden lukumäärä.

$$10 \leq x < 200, \tag{13}$$

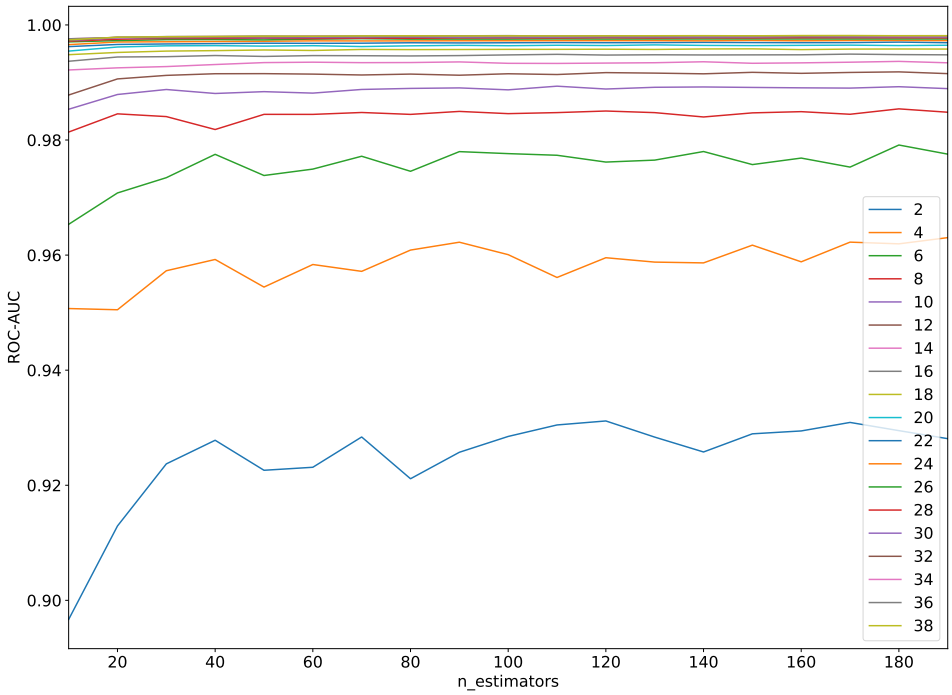
mutta nyt askellus on 10 eli arvoja valitaan 10 välein. Ensimmäinen arvo on tällöin 10, seuraava 20 ja niin edelleen.

Edellisten argumenttien lisäksi `GridSearchCV`[35]-luokalle annetaan `scoring`-argumenttina arvo `roc_auc`, mikä tarkoittaa sitä, että mallin hyvyyden arviointiin käytetään ROC-AUC-arvoa. Kuvassa 5.5 on esitetty mallin hyvyyden muutos (ROC-AUC) satunnaismetsän päätöspuiden määrään funktiona. Arvot on ryhmitelty päätöspuun maksimisyvyyden mukaan.

Kuvasta voidaan havaita, että puiden määrä vaikuttaa luokittelijan vakauteen: mitä enemmän metsässä on puita, sitä tasaisempi käyrä. ROC-AUC-arvoon puiden määrä vaikuttaa kuitenkin vähemmän. Esimerkiksi, kun puun syvyydeksi asetetaan kaksi ja puiden määrä nousee kymmenestä neljään kymmeneen, ROC-AUC-arvo nousee likimäärin kolme prosenttiyksikköä, mutta sen jälkeen puiden lisääminen ei enää vaikuta paljoa ROC-AUC-arvoon.

Kuvan perusteella puun syvyys näyttäisi olevan tärkeämpi parametri kuin puiden määrä. Mitä syvempi puu on, sitä vähemmän kuvassa näkyy vaihtelua ja ROC-AUC-arvokin on selkeästi parempi suuremmilla syvyyksillä. Maksimisyvyy-

[35] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html



Kuva 5.5: ROC-AUC-arvo ja päätöspuiden lukumäärä. Ryhmittely puun maksimisyvyyden mukaan.

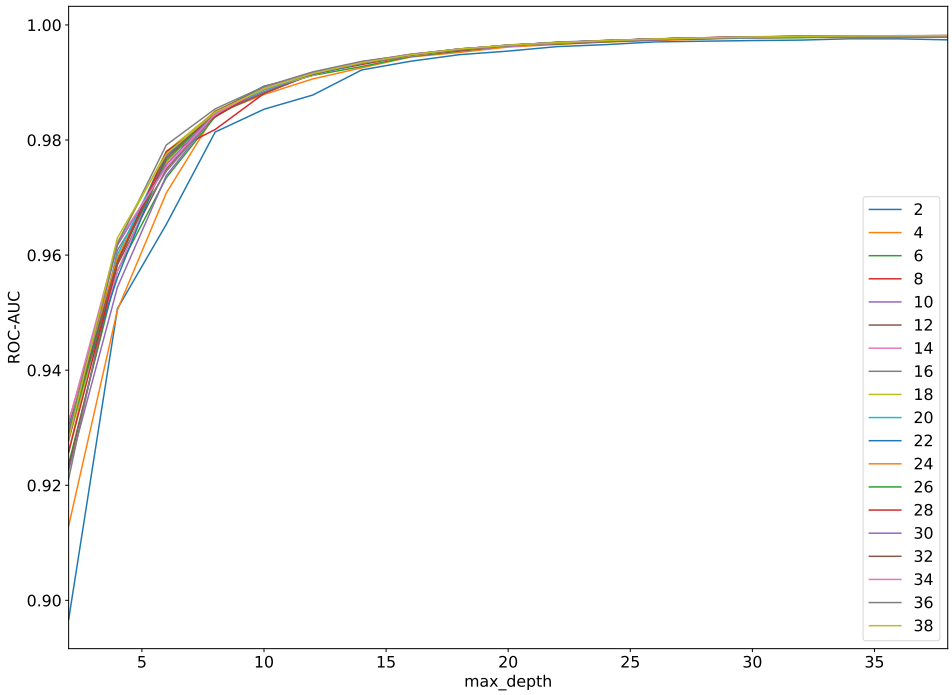
den lisäämisessä on kuitenkin nähtävissä samankaltainen ilmiö kuin puiden lisäämisessä: määrätyn arvon jälkeen ROC-AUC-arvon kasvu hidastuu voimakkaasti.

Kuvassa 5.6 esitetään kuvan 5.5 tilanne hieman toisesta näkökulmasta. Nyt ROC-AUC-arvo kuvataan maksimisyvyyden funktiona ja ryhmittely on tehty päätöspuiden lukumäärän mukaan. Kuvasta havaitaan sama tilanne kuin edellisestä kuvasta: ROC-AUC-arvo kasvaa aluksi nopeasti syvyyttä lisättäessä, mutta kasvu hidastuu voimakkaasti maksimisyvyyden saavutettua määrätyn arvon.

5.3.2 Muuttujien tärkeys

Satunnaismetsä, kuten myös yksittäinen päätöspuu, tuottaa arvot muuttujien tärkeydelle. (Louppe, 2014, ss. 123-125) Kuvassa 5.7 esitetään edellä kuvatun testiajon tuottaman satunnaismetsän antamista muuttujien tärkeysarvoista kaksikymmentä ensimmäistä.

Satunnaismetsän tärkeimmäksi määrittämä muuttuja, SYSTEM_ALERT_WINDOW-oikeus, kontrolloi ikkunoiden piirtämistä muiden sovellusten päälle. Aiemmin luvussa 2 kuvatut kiristysohjelmat voivat käyttää tätä oikeutta lukitsemaan laitteen niin, että käyttäjä ei pääse käsiksi laitteen toimintoihin ennen kuin, esimer-



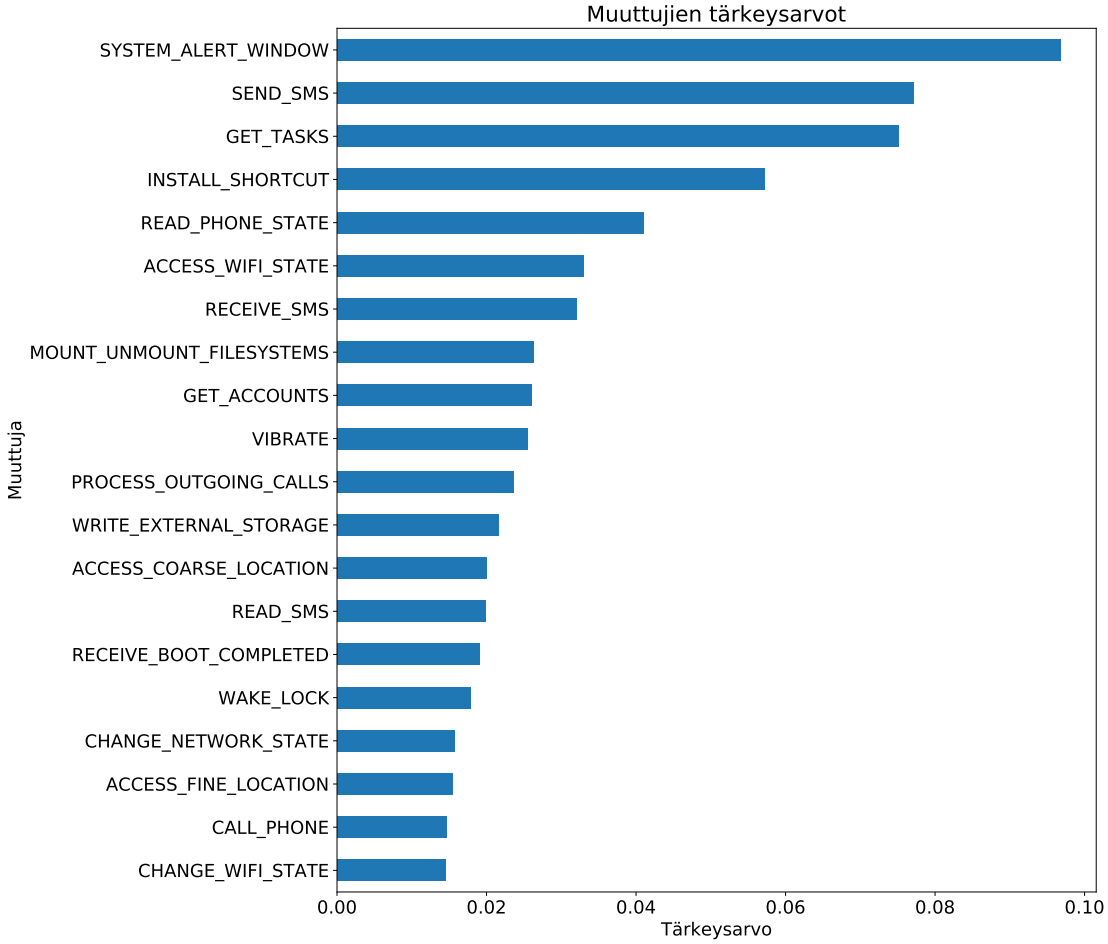
Kuva 5.6: `ROC-AUC`-arvo maksimisyvyyden funktiona. Ryhmittely päätöspuiden lukumäärän mukaan.

kiksi, maksaa kiristäjälle lunnaat. (Lindorfer ym., 2014, s. 7).

Lisäksi `SYSTEM_ALERT_WINDOW`-oikeutta voidaan käyttää esimerkiksi niin sanottuihin *napautuksen kaappaus* (tapjacking) -hyökkäyksiin, jossa ruudulle avataan jokin sovellusikkuna, mutta käyttäjän napautukset välittyvätkin alla olevalle sovellukselle. Ren ym. kuvaavat artikkelissaan Bankbot-haittaohjelman toimintaa. Bankbot piirtää ruudulle ikkunan, jonka alla on laitteen admin-oikeuksien hyväksyntäikkuna. Kun käyttäjä napauttaa päällimmäisen ikkunan OK-painiketta (tai vastaavaa), hän antaa tietämättään haittaohjelmalle admin-oikeudet. (Ren ym., 2017, s. 10; Lindorfer ym., 2014, s. 7)

Toisena tärkeydessä on `SEND_SMS`-oikeus. Tämä oikeus antaa sovellukselle kyvyn lähettää tekstiviestejä pyytämättä käyttäjältä siihen lupaa erikseen. Tätä oikeutta käyttävät sellaiset haittaohjelmat, jotka pyrkivät lähettämään tekstiviestejä maksullisiin palvelunumeroihin. (Dini ym., 2012, s. 6; Felt, Ha ym., 2012)

F-Securen (F-Secure, ei julkaisupäivää[b]) mukaan satunnaismetsän muuttujille antamat tärkeysarvot vaikuttavat olevan järkeviä tuloksia.



Kuva 5.7: Kaksikymmentä tärkeintä muuttujaa satunnaismetsän antamien tärkeysarvojen mukaan järjestettynä.

5.3.3 Liittyvä tutkimus

Sanz ym. (Sanz ym., 2013), joihin viitattiin jo *k*:n lähimmän naapurin menetelmän yhteydessä, käyttivät myös satunnaismetsää omassa tutkimuksessaan. He käyttivät tutkimuksessaan hyvin pientä datajoukkoa ja myös mainittuja eri puiden määriä oli vain kolme (10, 50 ja 100), kun tässä tutkielmassa käytettiin testiajoissa huomattavasti useampia eri vaihtoehtoja.

Sanz ym. käyttivät muuttujina muun muassa sovellusten vaatimia oikeuksia. Oikeuksien kohdalla he saivat `ROC-AUC`-arvoksi parhaimmillaan 0.95 puiden lukumäärän ollessa 100. Tässä tutkielmassa 100 päätöspuulla – maksimisyvyyden ollessa 38 – `ROC-AUC`-arvoksi saatiin 0.9980, mikä on huomattavasti parempi tulos. Eräs syy eroihin tuloksissa saattaa olla datajoukon koko: tässä tutkielmassa käytetty datajoukko oli kooltaan lähes 200-kertainen verrattuna Sanz ym. käyttämään datajoukkoon.

Myös Alam ym. (Alam ja Vuong, 2015) sekä Ham ym. (Ham ym., 2013) käyttivät satunnaismetsää tutkimuksessaan. Alam ym. tasapainottivat datajoukkonsa

luokkajakaumaa keinotekoisilla tapauksilla. Ham ym. eivät käyttäneet oikeuksia muuttujina tutkimuksessaan, mutta saivat satunnaismetsällä hyviä tuloksia, ROC-AUC-arvon ollessa 0.998, mikä on lähellä tämän tutkielman parasta ROC-AUC-arvoa.

6 Yhteenveto

Edellä on tarkasteltu lyhyesti alan tutkimuksen lähtökohtia, haittaohjelmia muun muassa esimerkkien kautta, Android-käyttöjärjestelmää sekä Android-sovelluksia ja niiden turvallisuuskysymyksiä. Lisäksi luvuissa 4 ja 5 on esitelty käytännön tutkimuksessa käytetty datajoukko sekä menetelmät, vastaavasti. Tuloksia on vertailtu muuhun alan tutkimukseen.

Käytännön tutkimustyön tulokset näyttävät osoittavan, että Android-sovellusten vaatimia järjestelmäoikeuksia voidaan käyttää haittaohjelmien tunnistamisessa menestyksekkäästi suhteessa väärien positiivisten määrään. Järjestelmäoikeudet eivät kuitenkaan vaikuta tarjoavan riittävästi informaatiota haittaohjelmien tunnistamiseen, sillä – vaikka väärien positiivisten määrä saatiin laskettua alhaiselle tasolle – jäi huomattava määrä haittaohjelmista kuitenkin tunnistamatta.

Tutkielman suuntaan vaikuttaneessa Moonsamyn ym. (Moonsamy ym., 2014) tutkimuksessa käytettiin myös sovelluksen vaatimia järjestelmäoikeuksia osana muuttujajoukkoa. Tämän tutkielman datajoukossa hyvänlaatuisten sovellusten 20 yleisimmin vaatimaa oikeutta olivat lähes samat kuin Moonsamylla. Haitallisten sovellusten vastaavalla listalla taas oli enemmän eroavaisuuksia, joista huomattavin oli `SEND_SMS`-oikeuden puuttuminen. Tämä järjestelmäoikeus mahdollistaisi `SMS`-viestien lähettämisen maksullisiin numeroihin, mikä on eräs joidenkin haitallisten sovellusten toiminnallisuuksista (Dini ym., 2012, s. 6). Oikeuden puuttumista voisi mahdollisesti selittää alan nopea muutos: taloudellisen hyödyn tavoittelu lähettämällä `SMS`-viestejä maksullisiin numeroihin ei ole kovin vanha ilmiönä. Lisäksi Moonsamyn artikkeli on tätä kirjoitettaessa yli kaksi vuotta vanha, mikä on pitkä aika tällä alalla.

Käytetyistä menetelmistä `k`:n lähimmän naapurin luokittelumenetelmä osoittautui käytännössä liian hitaaksi ja raskaaksi henkilökohtaisella tietokoneella. Sen käyttö olisi kuitenkin mahdollista sellaisissa olosuhteissa, joissa on käytettävissä huomattava määrä laskentatehoa ja -kapasiteettia. Testiajoissa kokeiltiin myös ajatusta siitä, että kaikki uuden tapauksen naapureista olisi oltava haitallisia, jotta tapaus luokiteltaisiin haitalliseksi. Tulokset tällaisessa testitilanteessa olivat hyviä väärien positiivisten lukumäärään nähden.

Päätöspuu oli erittäin nopea ja tuotti hyviä tuloksia, mutta sen epävakauden vuoksi suositeltavampaa olisi käyttää vakaampaa satunnaismetsää. Satunnaismetsän avulla saatiinkin hieman parempia tuloksia kuin yksittäistä päätöspuuta käytettäessä. Satunnaismetsästä saatavat muuttujien tärkeysarvot vaikuttivat vastaavan todellisuutta.

Kirjalliset viitteet

Alam, Mohammed S. ja Son T. Vuong (2015). ”Performance of malware classifier for android”. Teoksessa: s. 1–7. DOI: 10.1109/IEMCON.2015.7344482 (ks. s. 47).

Breiman, Leo (1996). ”Bagging Predictors”. *Machine Learning* 24.2, s. 123–140. DOI: 10.1007/BF00058655 (ks. s. 42).

— (2001). ”Random Forests”. *Machine learning* 45.1, s. 5–32. DOI: 10.1023/A:1010933404324 (ks. s. 30, 37, 42).

Breiman, Leo ym. (1984). *Classification and Regression Trees*, s. 368 (ks. s. 37, 38).

Burguera, Iker ym. (2011). ”Crowdroid: Behavior-Based Malware Detection System for Android”. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, s. 15. DOI: 10.1145/2046614.2046619 (ks. s. 6, 25).

Cyber Threat Alliance (2015). ”Lucrative Ransomeware Attacks: Analysis of the CryptoWall Version 3 Threat”, s. 59 (ks. s. 12, 14).

Dini, Gianluca ym. (2012). ”MADAM: a Multi-Level Anomaly Detector for Android Malware - Technical Report”. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7531 LNCS, s. 240–253. DOI: 10.1007/978-3-642-33704-8-21 (ks. s. 46, 49).

Drake, Joshua J ym. (2014). *Android Hacker's Handbook*. Wiley. ISBN: 9781118922255 (ks. s. 15–17, 20).

Duda, Richard O. ym. (2001). *Pattern Classification*, s. 680. ISBN: 0471223611 (ks. s. 37).

Elenkov, Nikolay (2014). *Android Security Internals : An In-Depth Guide to Android's Security Architecture*. No Starch Press. ISBN: 9781593276416 (ks. s. 15–20, 22).

Elkan, Charles (2011). ”Nearest Neighbor Classification”, s. 16 (ks. s. 29, 30).

Felt, Adrienne Porter, Erika Chin ym. (2011). ”Android permissions demystified”. Teoksessa: s. 627–637. DOI: 10.1145/2046707.2046779 (ks. s. 20).

Felt, Adrienne Porter, Matthew Finifter ym. (2011). ”A survey of mobile malware in the wild”. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, s. 3–14. DOI: 10.1145/2046614.2046618 (ks. s. 6, 8, 12).

Felt, Adrienne Porter, Elizabeth Ha ym. (2012). ”Android permissions: User Attention, Comprehension, and Behavior”. Teoksessa: DOI: 10.1145/2335356.2335360 (ks. s. 22, 46).

Flach, Peter (2012). *Machine Learning The Art and Science of Algorithms that Make sense of data*. ISBN: 9781107096394 (ks. s. 38).

F-Secure (2015b). ”Threat Report 2015” (ks. s. 9, 12, 14).

F-Secure (2016b). ”F-Secure DeepGuard: Proactive on-host protection against new and emerging threats”, s. 1–8 (ks. s. 6, 10).

Geurts, Pierre (2002). ”Contributions to Decision Tree Induction: Bias / Variance Tradeoff and Time Series Classification”. PhD Thesis. URL: http://hdl.handle.net/2268/25737 (ks. s. 36, 37).

Grzonkowski, Slawomir ym. (2014). ”Smartphone security: An overview of emerging threats”. 3.4, s. 40–44. DOI: 10.1109/MCE.2014.2340211 (ks. s. 12).

Ham, Hyo-sik ym. (2013). ”Analysis of Android malware detection performance using machine learning classifiers”. *2013 International Conference on ICT Convergence (ICTC)*, s. 490–495. DOI: 10.1109/ICTC.2013.6675404 (ks. s. 47).

Han, Jiawei ym. (2006). *The Morgan Kaufmann Series in Data Management Systems : Data Mining, Southeast Asia Edition : Concepts and Techniques (2)*. Morgan Kaufmann. ISBN: 9780080475585 (ks. s. 29).

Hastie, Trevor ym. (2009). ”The Elements of Statistical Learning”. *The Mathematical Intelligencer* 27.2, s. 745. DOI: 10.1007/b94608 (ks. s. 29).

James, Gareth ym. (2013). *An Introduction to Statistical Learning With Applications in R*, s. 618. ISBN: 978-1-4614-7137-0. DOI: 10.1007/978-1-4614-7138-7 (ks. s. 29, 42).

Kluyver, Thomas ym. (2016). ”Jupyter Notebooks—a publishing format for reproducible computational workflows”. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, s. 87–90. DOI: 10.3233/978-1-61499-649-1-87 (ks. s. 26).

Kohavi, Ron ja Chia-Hsin Li (1995). ”Oblivious Decision Trees, Graphs, and Top-Down Pruning”. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, s. 1071–1077 (ks. s. 37).

Koivisto, Pertti ja Riitta Niemistö (2001). ”Graafiteoriaa” (ks. s. 37).

Lindorfer, Martina ym. (2014). ”ANDRUBIS: 1,000,000 apps later: A view on current android malware behaviors”. *Proceedings of the International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, s. 8. DOI: 10.1109/BADGERS.2014.7 (ks. s. 46).

Louppe, Gilles (2014). ”Understanding Random Forests: From Theory to Practice”. Tohtorinväitöskirja, s. 223. DOI: 10.13140/2.1.1570.5928 (ks. s. 29, 36–38, 43, 45).

Mohri, Mehryar ym. (2014). *Foundations of Machine Learning*. MIT Press. ISBN: 9780262305662 (ks. s. 37).

Moonsamy, Veelasha ym. (2014). ”Mining permission patterns for contrasting clean and malicious android applications”. *Future Generation Computer Systems* 36, s. 122–132. DOI: 10.1016/j.future.2013.09.014 (ks. s. 6, 17, 20, 25, 49).

Quinlan, John Ross (1986). "Induction of Decision Trees". *Machine Learning* 1.1, s. 81–106. DOI: 10.1023/A:1022643204877 (ks. s. 37).

— (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers. ISBN: 9781558602380 (ks. s. 37).

Raileanu, Laura Elena ja Kilian Stoffel (2004). "Theoretical Comparison between the Gini Index and Information Gain Criteria". *Annals of Mathematics and Artificial Intelligence* 41.1, s. 77–93. DOI: 10.1023/B:AMAI.0000018580.96245.c6 (ks. s. 37).

Ren, Chuangang ym. (2017). "WindowGuard: Systematic Protection of GUI Security in Android". *Ndss* March. DOI: 10.14722/ndss.2017.23529 (ks. s. 46).

Rokach, Lior ja Oded Maimon (2007). *Data Mining with Decision Trees*. World Scientific Publishing Company. ISBN: 9789812771728 (ks. s. 37, 41, 42).

Sanz, Borja ym. (2013). "MAMA: Manifest Analysis for Malware Detection in Android". *Cybernetics and Systems* 44.6-7, s. 469–488. DOI: 10.1080/01969722.2013.803889 (ks. s. 34, 47).

Shannon, Claude Edmund (1948). "A Mathematical Theory of Communication". *Bell System Technical Journal* 5.3, s. 3. DOI: 10.1002/j.1538-7305.1948.tb01338.x (ks. s. 38).

Sikorski, Michael ja Andrew Honig (2012). *Practical Malware Analysis*. No Starch Press. ISBN: 9781593274306 (ks. s. 8).

Suarez-Tangil, Guillermo, Juan E. Tapiador ym. (2014). "Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families". *Expert Systems with Applications* 41.4 PART 1, s. 1104–1117. DOI: 10.1016/j.eswa.2013.07.106 (ks. s. 6).

Suarez-Tangil, Guillermo, Juan Tapiador ym. (2014). "Evolution, Detection and Analysis of Malware for Smart Devices". 16.2 (ks. s. 8, 11).

Wei, Xuetao ym. (2012). "Permission Evolution in the Android Ecosystem". *ACSAC '12 Proceedings of the 28th Annual Computer Security Applications Conference* April 2009, s. 31–40. DOI: 10.1145/2420950.2420956 (ks. s. 23).

Viestintävirasto (2015c). "Viestintäviraston Kyberturvallisuuskeskuksen vuosiraportti 2015", s. 1–25 (ks. s. 9, 11).

Yang, Chao ym. (2014). "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in android applications". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8712 LNCS.PART 1, s. 163–182. DOI: 10.1007/978-3-319-11203-9_10 (ks. s. 25).

Yerima, Suleiman Y ym. (2014). "Android Malware Detection Using Parallel Machine Learning Classifiers". *2014 Eighth International Conference on Next Genera-*

tion Mobile Apps, Services and Technologies, s. 37–42. DOI: 10.1109/NGMAST.2014.23 (ks. s. 41).

Zighed, D A ja R Rakotomalala (2000). *Graphes d'induction: apprentissage et data mining*. Hermes Science Publications. ISBN: 9782746200722 (ks. s. 37).

Sähköiset viitteet

contributors, Wikipedia (2017a). *Trojan horse (computing)*. URL: https://en.wikipedia.org/w/index.php?title=Trojan_horse_(computing) (viitattu 19. 06. 2017) (ks. s. 9).

— (2017[b]). *Android application package*. URL: https://en.wikipedia.org/w/index.php?title=Android%7B%5C_%7Dapplication%7B%5C_%7Dpackage%7B%5C&%7Doldid=782491961 (viitattu 14. 06. 2017) (ks. s. 17).

— (2016[a]). *Android (operating system)*. URL: https://en.wikipedia.org/w/index.php?title=Android%7B%5C_%7D(operating%7B%5C_%7Dsystem) (viitattu 01. 06. 2016) (ks. s. 15–17).

— (2017[c]). *Android Runtime*. URL: https://en.wikipedia.org/w/index.php?title=Android%7B%5C_%7DRuntime%7B%5C&%7Doldid=721696304 (viitattu 14. 06. 2017) (ks. s. 17).

— (2017[d]). *Ransomware*. URL: https://en.wikipedia.org/wiki/Ransomware (viitattu 18. 03. 2017) (ks. s. 9, 13).

— (2017[e]). *Sideloading*. URL: https://en.wikipedia.org/w/index.php?title=Sideloading%7B%5C&%7Doldid=780233157 (viitattu 14. 06. 2017) (ks. s. 17).

— (2016[b]). *Stagefright (bug)*. URL: https://en.wikipedia.org/w/index.php?title=Stagefright%7B%5C_%7D(bug)%7B%5C&%7Doldid=714720452 (viitattu 28. 06. 2016) (ks. s. 11).

— (2016[c]). *Trojan Horse*. URL: https://en.wikipedia.org/w/index.php?title=Trojan_Horse (viitattu 20. 11. 2016) (ks. s. 9).

Dormann, Will (2016). *Who Needs to Exploit Vulnerabilities When You Have Macros?* URL: https://insights.sei.cmu.edu/cert/2016/06/who-needs-to-exploit-vulnerabilities-when-you-have-macros.html (viitattu 26. 05. 2017) (ks. s. 11).

Ducklin, Paul (2013). *CryptoLocker ransomware - see how it works, learn about prevention, cleanup and recovery*. URL: https://nakedsecurity.sophos.com/2013/10/18/cryptolocker-ransomware-see-how-it-works-learn-about-prevention-cleanup-and-recovery/ (viitattu 20. 06. 2016) (ks. s. 13, 14).

F-Secure (2015a). *Android's Stagefright bug – phone vendors taken with their pants down*. URL: http://safeandsavvy.f-secure.com/2015/07/30/androids-

stagefright-bug-phone-vendors-taken-with-their-pants-down/ (viitattu 28. 06. 2016) (ks. s. 11, 12).

F-Secure (2016a). *Classification*. URL: https://www.f-secure.com/en/web/labs%7B%5C_%7Dglobal/classification (viitattu 15. 06. 2016) (ks. s. 8).

— (2016c). *What's the Deal With Machine Learning?* URL: https://labsblog.f-secure.com/2016/08/26/whats-the-deal-with-machine-learning/ (viitattu 20. 12. 2016) (ks. s. 6).

— (ei julkaisupäivää[a]). *Don't get taken! How to avoid ransomware like CTB-Locker*. URL: https://safeandsavvy.f-secure.com/2015/02/09/dont-get-taken-how-to-avoid-to-avoid-ransomware-like-ctb-locker/ (ks. s. 13).

— (ei julkaisupäivää[b]). *Personal conversations with F-Secure* (ks. s. 22, 32, 46).

— (2017). *Secure your smartphones and tablets*. URL: https://www.f-secure.com/en/web/home_global/mobile-security (viitattu 16. 03. 2017) (ks. s. 10).

— (2016[d]). *Terminology: An A-Z guide to the technical terms used in Labs*. URL: https://www.f-secure.com/en/web/labs%7B%5C_%7Dglobal/terminology (viitattu 18. 12. 2016) (ks. s. 6).

— (2016[e]). *Trojan: Android/DroidDream.A Description*. URL: https://www.f-secure.com/v-descs/trojan%7B%5C_%7Dandroid%7B%5C_%7Ddroiddream%7B%5C_%7Da.shtml (viitattu 30. 05. 2016) (ks. s. 12).

— (2016[f]). *Trojans*. URL: https://www.f-secure.com/en/web/labs%7B%5C_%7Dglobal/trojans (viitattu 22. 06. 2016) (ks. s. 9).

— (2016[g]). *What's the Deal With Scanning Engines?* URL: https://labsblog.f-secure.com/2016/05/17/whats-the-deal-with-scanning-engines/ (viitattu 21. 06. 2016) (ks. s. 6, 10).

Google (2015). *Android 6.0 "Marshmallow"home screen*. URL: https://commons.wikimedia.org/w/index.php?curid=42781988 (ks. s. 15).

— (2017a). *ART and Dalvik*. URL: https://source.android.com/devices/tech/dalvik/ (viitattu 20. 06. 2017) (ks. s. 17).

— (2017[b]). *Requesting Permissions*. URL: https://developer.android.com/guide/topics/permissions/requesting.html (viitattu 31. 05. 2017) (ks. s. 18).

Kaspersky (2009). *Drive-by Downloads. The Web Under Siege*. URL: https://securelist.com/analysis/publications/36245/drive-by-downloads-the-web-under-siege/ (viitattu 23. 03. 2017) (ks. s. 8, 9).

— (2016a). *Antivirus fundamentals: Viruses, signatures, disinfection*. URL: https://blog.kaspersky.com/signature-virus-disinfection/13233/ (viitattu 17. 12. 2016) (ks. s. 6).

— (2017[a]). *Learn about malware and how to protect all your devices against it*. URL: https://www.kaspersky.com/resource-center/preemptive-safety/what-is-malware-and-how-to-protect-against-it (viitattu 24. 07. 2017) (ks. s. 8).

Kaspersky (2016[b]). *Ransomware Decryptor*. URL: https://noransom.kaspersky.com/faq-en/ (viitattu 20. 12. 2016) (ks. s. 13, 14).

— (2016[c]). *What is a Macro Virus?* URL: https://usa.kaspersky.com/internet-security-center/definitions/macro-virus (viitattu 21. 12. 2016) (ks. s. 11).

— (2017[b]). *Why you should NOT pay ransom to malware creators*. URL: https://blog.kaspersky.com/no-no-ransom/13364/ (viitattu 18. 03. 2017) (ks. s. 13).

Microsoft (2011). *What You Should Know About Drive-By Download Attacks – Part 1*. URL: https://blogs.microsoft.com/microsoftsecure/2011/12/08/what-you-should-know-about-drive-by-download-attacks-part-1/ (viitattu 20. 06. 2017) (ks. s. 9).

— (ei julkaisupäivää). *Enable or disable macros in Office files*. URL: https://support.office.com/en-us/article/Enable-or-disable-macros-in-Office-files-12b036fd-d140-4e74-b45e-16fed1a7e5c6 (ks. s. 10).

— (2017). *New feature in Office 2016 can block macros and help prevent infection*. URL: https://blogs.technet.microsoft.com/mmpc/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/ (viitattu 17. 03. 2017) (ks. s. 10, 11).

NBC (2016). *LA Hospital Paid 17K Ransom to Computer Hackers | NBC Southern California*. URL: http://www.nbclosangeles.com/news/local/Hollywood-Presbyterian-Paid-17K-Ransom-to-Hackers-369199031.html (viitattu 22. 06. 2016) (ks. s. 14).

Rissanen, Juha ja Esa Koivuranta (2016). *Verkkorikolliset tunkeutuvat sairaalan verkkoon, lukitsevat tiedostoja ja vaativat rahaa – Ovatko tietoni turvassa?* URL: http://yle.fi/uutiset/verkkorikolliset_tunkeutuvat_sairaalan_verkkoon_lukitsevat_tiedostoja_ja_vaativat_rahaa__ovatko_tietoni_turvassa/8904018 (viitattu 20. 06. 2016) (ks. s. 8, 14).

Scikit Learn (2016). *Decision Trees (User Guide)*. URL: http://scikit-learn.org/stable/modules/tree.html (viitattu 27. 12. 2016) (ks. s. 37, 39).

Sophos (2016). *Ransomware that's 100% pure JavaScript, no download required*. URL: https://nakedsecurity.sophos.com/2016/06/20/ransomware-thats-100-pure-javascript-no-download-required/ (viitattu 28. 06. 2016) (ks. s. 11).

Trend Micro (2013). *World Backup Day: The 3-2-1 Rule*. URL: http://blog.trendmicro.com/trendlabs-security-intelligence/world-backup-day-the-3-2-1-rule/ (viitattu 19. 06. 2017) (ks. s. 13).

— (2017). *Ransomware*. URL: https://www.trendmicro.com/vinfo/us/security/definition/Ransomware (viitattu 18. 03. 2017) (ks. s. 13).

Wei, Wan (2016). *Interview with Mikko Hypponen, CRO of F-Secure*. URL: https://thehieno.com/2016/12/18/the-hieno-suomi-100-series-interview-with-mikko-hypponen-cro-of-f-secure/ (viitattu 22. 12. 2016) (ks. s. 11).

Viestintävirasto (2015a). *Exploit kit - tehokas haittaohjelmien levittäjä*. URL: https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvanyt/2015/03/ttn201503061108.html (viitattu 27. 06. 2016) (ks. s. 9).

— (2015b). *Haittaohjelmien toimintaperiaatteet, tarttuminen ja niiltä suojautuminen*. URL: https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvanyt/2015/02/ttn201502101246.html (viitattu 20. 06. 2017) (ks. s. 8, 11).